VŠB – Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science

MASTER THESIS

2010 Adam Farnik

VŠB – Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science

Possibilities of employing ASP.NET technology in Linux Environment

Možnosti používání technologie ASP.NET v prostředí Linuxu

2010 Adam Farnik

Declaration

Hereby I declare that this bachelor/master thesis was written on my own.

I have quoted all references I have drawn upon

Date …..... Signature….....

Acknowledgement

I would hereby like to thank all who supported me during the work on this thesis. I would like to begin with the head of the thesis, Ing. Bogdan Siderek, and thank him for his consultations regarding the thesis. I would also like to thank Dr. Tzong-An Su, Associate Professor of Department of Information Engineering and Computer Science on Feng Chia University in Taichung, Taiwan, for his efforts to provide me with the needed equipment. And finally I would like to thank to people working in the Office of Information Technology on aforementioned Feng Chia University, whom I have shared the office with and who have been very helpful in numerous ways during the period I was working there on this thesis.

Abstract

This master thesis describes the differences between Microsoft Windows based hosting environment for ASP.NET web pages, applications and services, and its open source alternative implementation installed in Linux OS. Starting from the installation of both operating systems and the necessary software, it continues with the development of the information system implemented in ASP.NET, its porting from Windows to Linux environment and finishes with comparison of efficiency of both solutions. The goal of this paper is to present the alternatives compared in a specific scenario.

Keywords

ASP.NET, Mono, OpenSUSE, Apache, MySQL, Windows Web Server 2008, MS SQL, IIS, efficiency comparison, information system

Abstrakt

Tato diplomová práce popisuje rozdíly mezi hostingem ASP.NET webových stránek, aplikací či služeb na platformě Microsoft Windows a jeho open source alternativní implementací instalovanou v prostředí Linuxového OS. Počínaje popisem instalace obou operačních systému a dalšího nezbytného software, pokračuje vývojem ASP.NET informačního systému, jeho portování z prostředí Windows do Linuxu a konče na srovnání efektivity obou řešení. Cílem, který si tato práce klade je předvést srovnání dvou alternativ na konkretním příkladě.

Klíčová slova

ASP.NET, Mono, OpenSUSE, Apache, MySQL, Windows Web Server 2008, MS SQL, IIS, srovnání efektivity, informační systém

List of used abbreviations

ADO.NET	Active Data Object .NET
ASP.NET	Active Server Pages .NET
Auto-MDIX	Automatic Medium-Dependent Interface Crossover
BLL	Business Logic Layer
CGI	Common Gateway Interface
CPU	Central Processing Unit
DAL	Data Access Layer
DLL	Dynamic-link Library
ECMA	European Computer Manufacturers Association
FastCGI	Fast Common Gateway Interface
GDI+	Graphics Device Interface
GRUB	Grand Unified Bootloader
HTML	HyperText Markup Language
IIS	Internet Information Services
ISO/IEC	International Organization for Standardization/International Electrotechnical Comision
LVM	Logical Volume Manager
ODBC	Open Database Connectivity
SDK	Software Development Kit
TCP	Transmission Control Protocol
TUI	Text User Interface
URL	Unified Resource Locator
UTP	Unshielded Twisted Pair
XML	Extensive Markup Language

Table of contents

1 Introduction

ASP.NET is a web application framework increasingly popular in the world of web design. It has been developed by Microsoft as a part of .NET Framework, where it is a part of the Framework Class Library. Ever since the C# programming language specification and .NET Framework's Common Language Interface, providing virtual machine enabling the code to be run on different architectures without the need to be rewritten, were ratified by ECMA and later also ISO/IEC standards (namely in ECMA-334, ECMA-335 and ISO/IEC 23270:2006, ISO/IEC 23271:2006), there has been much effort to create alternative implementations of .NET Framework.

The most important goal of one of those implementations – Mono, is to provide the .NET Framework functionality also on different operating systems, not only Microsoft Windows. Mono is the open-source project, started by Ximian and currently developed by Novell, that can be run on Linux, BSD, UNIX, Mac OS X, Solaris and Windows operating systems.

Although aforementioned ECMA and ISO/IEC standards do not cover such parts of .NET Framework as Windows Forms, ADO.NET, nor ASP.NET, they have been (along with others) implemented in Mono to some extent. In fact, mono developers claim, that they have completed the implementation of Windows Forms 2.0, ASP.NET 2.0 is almost completely covered and so is its extension ASP.NET AJAX. Furthermore, there are many classes implemented in Mono Class Library, that mostly provide functionality for building Linux applications, that are not present in Microsoft's .NET Framework implementation.

There are many ways to host an ASP.NET web page using Mono and it is possible on all the platforms, that Mono can be run at. It supports Apache web server (via mod_mono module), FastCGI-based web servers and CGI-based web servers (via cgi-fcgi bridge) [1]. Mono also has many ADO.NET data providers allowing to connect to all major database systems, including Microsoft SQL Server, MySQL, PostgreSQL, Oracle, etc., and ODBC data provider [2].

With these two alternatives of ASP.NET web pages hosting outlined, an obvious question comes about efficiency and usability. This thesis is focusing on comparison of these two methods, providing an overview on their effectivity.

The comparison itself was carried out on the same ASP.NET web page hosted in the first case by Microsoft IIS 7.0 on Windows Web Server 2008 (further referred to simply as *Windows*) connecting to Microsoft SQL Server 2008 database and in the second case by Apache 2.2.10 web server with mod_mono module on OpenSUSE 11.1 (further referred to simply as *Linux*) connecting to MySQL 5.0.62 database.

There is of course no way to investigate the efficiency in all possible scenarios,

therefore this thesis can not provide any definitive answer. Its goal is to provide an overlook allowing to understand how developed the Mono project is, and possibly facilitate the decision which solution to choose in particular case. The fact, that different operating system and database system are used in each case, makes the comparison to be more solution-oriented, rather then .NET implementation oriented.

Since the topic of this thesis is closely related to the controversy between Windows vs. Unix platform, as well as open source vs. non-open source software, I would like to emphasize, that the topic was approached without prejudice and the aim of the comparison is not to defend one kind of software or platform in favour of the other.

2 Hardware

In order to provide the most realistic testing environment a real computer was used instead of a virtual machine. The performance of virtual machine can be affected by the workload of hosting computer, so the difference between two tests run at a different time can not be avoided. It is also possible, that specific virtual machine may be optimized for one system and not the other.

The web server computer (further referred to as *server*) used for the tests was Hewlett-Packard HP d530 CMT (DC577AV). Its specification can be found in Table 1 below.

Component	**Type**
CPU	Intel Pentium 4, Prescott, 3000MHz
RAM	1536MB DDR-400 (10MB of which is used by GPU)
Chipset	Intel 865G Sprigngdale (82865G)
GPU	AGP 8x integrated in Intel 82865G (using 10MB RAM)
HDD	Western Digital 80GB, 7200 rpm, IDE (WD800BB-22HEA1)
Network adapter	Broadcom NetXtreme Gigabit Ethernet

Table 1: Server hardware

Apart from the web server, there was another computer used, from where the tests were run (further referred to as *client*). Connection to the server was realized by Ethernet, directly (without use of switch, hub or any gateway) with UTP cable (non-crossover, the ports were automatically configured by Auto-MDIX). Client computer was a Lenovo IBM ThinkPad R61 laptop with configuration as follows in Table 2.

Component	**Type**
CPU	Intel Core 2 Duo T7100, 1,8GHz
RAM	2048MB DDR2-667
Chipset	Intel PM965 Crestline (82965PM)
HDD	Hitachi 320GB, 7200rpm, SATA II (HTS723232L9A360)
Network adapter	Intel 82566MC Gigabit Ethernet

Table 2: Client hardware

3 Installation

Both operating systems, in which the hosting was provided, were installed on the server machine at the same time. In this chapter the installation and configuration process is described. Since the installation of ASP.NET hosting environment on Windows platform is very straightforward, the accent was put on the description of installation in Linux environment, so it could be used as a guide to this process.

3.1 Linux environment

This sub-chapter is the step-by-step description of the installation and configuration process of the Linux web server capable of hosting ASP.NET web pages. The procedure was based on the online article on the server HowtoForge [3], but was modified to serve the needed purpose.

3.1.1 OpenSUSE 11.1 installation

There were multiple reasons why to choose OpenSUSE Linux distribution. Most importantly, it is supported by Novell, just as the Mono project is, which promises a good compatibility. Furthermore, OpenSUSE is a distribution with a long tradition and is known for its stability. And unlike SUSE Linux Enterprise server it is freely available.

At the time of installing OpenSUSE 11.1 was the latest stable release, thus it was the version used. The installation medium was obtained from OpenSUSE official web page in a form of DVD ISO image for i586 architecture.

3.1.1.1 General setup

After booting from the DVD medium, the installation was started with the standard boot options. Following steps, such as selecting language, keyboard layout or time zone, are guided by the installation GUI. The installation mode was set to *New Installation* with the option of *Use Automatic Configuration* chosen.

In the Desktop Selection screen the *Minimal Server Selection (Text Mode)* option was chosen. That means that no X Window System components were installed and server was managed from the shell only. As a result, less system resources are used and thus available for the web server or database applications.

3.1.1.2 Hard drive partitioning

To install both Windows and Linux operating systems on the same hard drive, at least 3

partitions are needed – one Linux swap partition, one Linux root partition and one Windows partition (Windows uses swap file instead of swap partition). Even though a separate partition with /srv mountpoint was created for the web page source, it was still possible to create all partitions as primary. This scheme was used for simplicity in described comparison scenario (for the real web server, the use of LVM partitioning scheme would be advisable as it provides more complexity).

The partitioning scheme of the hard drive is to be found in Table 3 (size is in megabytes, with 1 MB = 1 048 576 B).

Partition	**Format**	**Mountpoint**	**Size**
sda1	swap		2 996 MB
sda2	ext3	/	12 092 MB
sda3	ext3	/srv	15 118 MB
sda4	NTFS		45 670 MB

Table 3: Partitioning of /dev/sda drive

OpenSUSE, just as any other Linux distribution, does not support NTFS file system by default. Therefore partition sda4 was just created, not formatted, at the time of OpenSUSE installation. It was formatted to NTFS at the time of installing Windows.

3.1.1.3 Creating user

In the next step, setup required to create new user. The only user created had username "*admin*" and the options of *using its password for system administrator* and *receiving system mail* enabled, whereas *autologin* option was disabled. First of mentioned options assigns the same password as filled for user "*admin*" to user "*root*".

At this point system was ready to be installed. Once done, system rebooted and performed automatic configuration.

3.1.2 Configuration

In OpenSUSE, even if no graphical interface is installed, some configuration can be done in two ways – either via TUI or configuration files. In this document the latter is preferred. All following procedures are done under the *root* account.

3.1.2.1 Network

In described scenario, server was connected to network via Ethernet interface *eth0*

having a public IP address statically assigned. In order to configure network connection *vim* editor was used and following configuration files were edited as shown:

```
# vim /etc/sysconfig/network/ifcfg-eth0

      IPADDR='140.134.23.133/24'
      BOOTPROTO='static'
      STARTMODE='auto'

# vim /etc/sysconfig/network/routes

      default 140.134.23.254 - -

# vim /etc/HOSTNAME

      webserver.fcu.edu.tw

# vim /etc/hosts

      127.0.0.1  localhost webserver.fcu.edu.tw webserver
      140.134.23.133  webserver.fcu.edu.tw  webserver

# vim /etc/resolv.conf

      search fcu.edu.tw
      nameserver 140.134.4.1

# /etc/init.d/network restart
```

The last command restarts the network interfaces and services, applying the new settings.

3.1.2.2 Firewall

Although the use of firewall was not absolutely necessary in this scenario, it was decided to configure it due to two reasons – firstly, the server was connected to the Internet for most of the time, except for the testing itself, and therefore securing it was advisable; and secondly, in real use firewall would most likely be set up, so it is more objective to let it affect the performance during the testing as well.

OpenSUSE installed in described scenario contained *iptables* firewall application. However it was controlled by the *yast2* TUI. Since approach taken in this paper is configuration files oriented, the firewall in *yast2* was disabled – *yast2* was executed from the shell, in menu *Security and Users* item *Firewall* was chosen and following two options were activated: *Disable Firewall Automatic Starting* and *Stop Firewall Now.*

Next the *iptables* commands were added to the */etc/init.d/boot.local* file:

```
iptables -P INPUT DROP
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -m state --state ESTABLISHED,RELATED \
     -j ACCEPT
```

The meaning of above rules is as follows: blocking all incoming traffic; allowing all incoming connections to loopback interface; allowing TCP connections to port 22 (allowing remote control via SSH); allowing TCP connections to port 80 (allowing incoming HTTP request to the web server); allowing incoming packets and datagrams for the connections established by the server. The rules were effective since the next system boot-up.

The configuration was finished by refreshing repositories, updating installed system's components to the latest versions (both done with *zypper* package manager), applying needed patches (if any) and rebooting. That was done with commands as follows:

```
# zypper refresh
# zypper update
# zypper patch
# reboot
```

The kernel version after update was 2.6.27.45-0.1-pae.

3.1.3 Software

Software installation and configuration was done from the shell, just as the system configuration. All commands were run by user *root* and *zypper* package manager was used for installation.

3.1.3.1 Database system – MySQL

Mono can access numerous database systems – MySQL was chosen due to its popularity and because it is licensed as open source MySQL server (version 5.0.67), along with the standard client programs were installed by running command:

```
# zypper in mysql mysql-client
```

Next, the *mysql_secure_installation* script was run. It is a part of MySQL installation, that creates new password for root MySQL user, removes anonymous user accounts and testing database, and disables remote root login.

3.1.3.2 HTTP server – Apache

According to the February 2010 survey [4] more than a half of all domains are hosted

by Apache HTTP servers. Apache also is an open source software, and there is a mod_mono module available, that allows the HTTP server to cooperate with Mono in order to provide an ability to host ASP.NET web pages.

Apache by default installs with *prefork* module, which makes it a non-threading web server with each request being handled by one process. However, according to the *mod_mono* homepage [5], it is recommended to use the *worker* module when hosting ASP.NET web pages on Apache. The threading model implemented by that module works better with mono.

To install version 2.2.10 of Apache and configure its automatic start during boot up, following commands were executed:

```
# zypper in apache2 apache2-worker
# chkconfig -a apache2
```

The aforementioned *mod_mono* homepage also suggests to turn off Apache's *KeepAlive* ability as it often negatively affects performance when hosting ASP.NET web pages. Following that suggestion, following change was introduced to the */etc/apache2/server-tutning.conf* file which is being included in Apache's default configuration file – */etc/apache2/httpd.conf*:

```
KeepAlive Off
```

3.1.3.3 Mono and related packages

Information in this sub-chapter were also reformulated and used in the OpenSUSE documentation wiki as a how-to article [6].

Packages of Mono itself are a part of default repository *repo-oss*, however in order to be sure, that the latest stable packages were installed, the Novell's separate Mono repository was added. That was done by running following command:

```
# zypper ar -cfn mono-stable \
> http://ftp.novell.com/pub/mono/download-stable\
> /openSUSE_11.1 mono-stable
```

Several packages need to be installed in order to make Apache web server capable of serving ASP.NET pages. The *mono-core* package contains the .NET runtime. Another needed package is *xsp* – it contains a small http server written in C#, that can process ASP.NET web pages. While XSP server could be theoretically used instead of Apache, its configuration possibilities are very limited, and so some more developed web server, such as Apache, would be needed for a real use. The *apache2-mod_mono* package is the Apache module, that allows it to proxy the ASP.NET requests to XSP server. The last needed package is *libgdiplus0*. It is a library with implementation of Microsoft's GDI+. It is needed for some graphical page components, such as buttons.

As the default repositories contain outdated version of first three mentioned packages,

installation of those was done with specifying source repository, whereas the *libgdiplus0* package was installed separately from the default repository:

```
# zypper in --repo mono-stable mono-core xsp apache2-mod_mono
# zypper in libgdiplus0
```

Since only one web page was hosted, there was no need to use Apache's virtual hosts. Mod_mono configuration file is placed in */etc/apache2/conf.d* directory, from where it is loaded on Apache start-up (by the */etc/apache2/default-server.conf* included in Apache's default configuration file – */etc/apache2/httpd.conf*; *default-server.conf* by default includes all the configuration files in *conf.d* directory) and the only change done to it was adding the following line, setting Apache to direct its ASP.NET processing requests to *mod-mono-server2*, that implements version 2.0 of .NET, instead of default *mod-mono-server*, that implements version 1.1.

```
MonoServerPath "/usr/bin/mod-mono-server2"
```

3.2 Windows environment

During the installation of Windows environment most of the settings were left default and whole process was done using GUI wizard. Therefore there is no need for so detailed description as in case of Linux environment and this sub-chapter is not following all the steps taken, but rather focusing on the most important ones.

3.2.1 Windows Web Server 2008 installation and configuration

Windows Web Server 2008 operating system was installed on the fourth partition created during Linux environment installation. Since Windows operating systems can only be installed on active partitions and after Linux installation the second partition (the one containing the root of filesystem) was set as active, the last partition had to be set so. It was done using the *diskpart.exe* program from Windows installation medium recovery console.

Windows Server 2008 and Windows Server 2008 R2 can be installed as Server Core, which provides minimal GUI and only can be controlled from the command line locally, or via remote control tools. Server Core provides somewhat limited functionality promising reduced maintenance, higher security, reduced management and lower disk space requirement [7]. Though it would provide more similar environment to the one of Linux, which was installed without the X Window System, Core Server does not support installation of Microsoft SQL Server [8], and so Windows had to be installed in the standard mode including Windows Explorer.

The system was installed from the Windows Web Server 2008 SP1 DVD with the

default options, network connection was set up with exactly the same parameters as used in Linux (see section 3.1.2.1 Network). Installation source was obtained via MSDN Academic Alliance program.

There was no need to install extra web server, since IIS is included in Windows by default. The *KeepAlive* setting was turned off just as in case of Apache.

Windows Firewall settings were changed, so that only following exceptions were allowed:

- Core Networking
- World Wide Web Services (HTTP)

3.2.1.1 Updating system

At this point the system update should have been performed. Windows system can not install some major updates (such as Service Pack) if the system is not located on the active partition. In this scenario, the system update was performed after the boot manager configuration, which led to the system misbehaviour.

After Windows Update located Service Pack 2 downloaded it and installed, it asked for system restart. Upon booting up, it was finishing the installation when it signalized error with code 80242016, rolled back the installation and rebooted again. This situation repeated on further attempts and Microsoft did not provide any documentation on the related error code. Only experimentally it was found out that the problem is caused by Windows being installed not on active partition.

The Windows Update was also not very efficient – instead of downloading all necessary updates right after the installation, it did so in 3 batches requiring a reboot after each.

3.2.1.2 Boot manager configuration

Since there were 2 operating system installed, it was necessary to configure the boot manager software, in order to provide the choice of which one to boot up. It was decided to configure the boot manager on the Linux partition. The default boot manager installed with OpenSUSE is GRUB. In its standard configuration is defines 3 boot options – standard and "failsafe" boot of OpenSUSE, and booting from the floppy. In order to add the option for booting Windows, it's configuration had to be altered. First, the second partition (Linux root filesystem) was declared active again and OpenSUSE was booted. Then the */boot/grub/menu.lst* file was edited and following lines were appended:

```
title Windows Web Server 2008
      rootnoverify (hd0,3)
      chainloader +1
```

3.2.2 Microsoft SQL Server 2008

The installation source for SQL server was obtained via MSDN Academic Alliance, just as the Windows operating system. The Enterprise version was installed. Apart of SQL server engine itself, also Microsoft SQL Server Management Studio was installed.

During installation the Administrator account for Windows was assigned to be the administering account for SQL server as well, only with the rights to connect locally granted however.

After updating SQL server with Windows Update, its version was 10.0.2531.0.

4 Information system development

As stated in the introduction, the comparison of both hosting alternatives was made on the same web page. More specifically, it was a web information system based on a database with precisely the same data. This chapter is describing its implementation.

The information system was implemented in Microsoft Visual Web Developer 2008 Express Edition.

4.1 Database

As the basis for the information system the MySQL testing database called Employees was used. It is a non-trivial relational database with solid amount of table entries released under Creative Commons Attribution-Share Alike 3.0 Unported License [9]. At the time of implementation, the latest version of Employees database (1.0.6) was used. The *employees_db-dump-files* release contains scripts creating the database and filling it with data, as well as the testing scripts, that verify whether imported data are correct. The structure of database is shown in ER diagram – see Illustration 1.

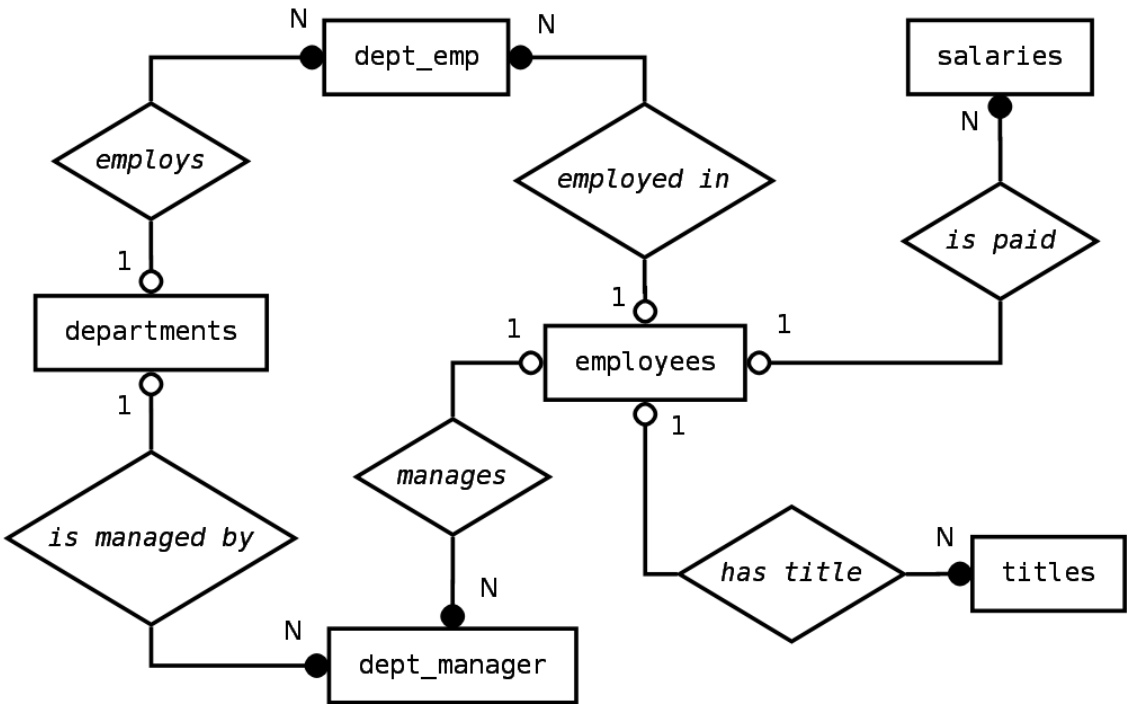


Illustration 1: ER Diagram - Employees database

Database engine type was set to InnoDB rather than MyISAM, as the second does not support transactions. Thought transactions were not used in the developed information system, the engine of Microsoft SQL is transactional, so it was decided to use transactional database engine for MySQL too, so that both engines would be comparable.

All the the references between the tables were defined with cascade referential action in case of deletion of the corresponding row in the master table.

Tables *employees* and *departments* are representing a simple lists with no foreign keys. The attribute names are self explanatory. The structure of the first mentioned is to be found in Table 4, whereas Table 5 describes the structure of the second.

Name	**Type**	**Size**	**NULL**	**Primary key**	**Index**	**Integrity constraints**
emp_no	INT	4B	N	Y	Y	auto-increment
birth_date	DATETIME	8B	N	N	N	
first_name	VARCHAR	14B	N	N	N	
last_name	VARCHAR	16B	N	N	N	
gender	ENUM	1B	N	N	N	“M”, “F”
hire_date	DATETIME	8B	N	N	N	

*Table 4: Structure of table **employees***

Name	**Type**	**Size**	**NULL**	**Primary key**	**Index**	**Integrity constraints**
dept_no	CHAR	4B	N	Y	Y	
dept_name	VARCHAR	40B	N	N	Y	

*Table 5: Structure of table **departments***

Follows the description of two tables used as a mediating tables to N:M relations between tables *departments* and *employees*. Table *dept_manager* (see Table 6) is holding the entries of the employees that are managing or used to manage some department and *dept_emp* (see Table 7) is a table assigning the employees to work in particular departments.

In both tables attributes *dept_no* are foreign keys from table *departments*, while attributes *emp_no* are foreign keys from table *employees*. Attributes *from_date* and *to_date* are defining the time period in which the employee was related with the department (as a manager or an employee respectively). If the relation of employee to the department is valid, the corresponding *to_date* attribute is set to Jan 1st 9999. In case of cancelling employee's managing or employment relation with the department, the corresponding row is not deleted, but the *to_date* attribute is set to the needed date.

While setting the date to Jan 1st 9999 for valid relations is not the most effective, it was taken from the original database design, as optimization of database is not the major concern of this thesis.

Name	Type	Size	NULL	Primary key	Index	Integrity constraints
dept_no	CHAR	4B	N	Y	Y	
emp_no	INT	4B	N	Y	Y	
from_date	DATETIME	8B	N	N	N	
to_date	DATETIME	8B	N	N	N	

*Table 6: Structure of table **dept_manager***

Name	Type	Size	NULL	Primary key	Index	Integrity constraints
dept_no	CHAR	4B	N	Y	Y	
emp_no	INT	4B	N	Y	Y	
from_date	DATETIME	8B	N	N	N	
to_date	DATETIME	8B	N	N	N	

*Table 7: Structure of table **dept_emp***

The last two tables are 1:N related to *employees* table with atribute *emp_no* being a foreign key in both cases. In table *titles* (see Table 8), the history of titles assigned to the employee is stored, whereas table *salaries* (see Table 9)is used to store the history of employee's salaries. In both cases the *from_date* and *to_date* attributes are used just as in the mediating tables *dept_manager* and *dept_emp* described above, with date of Jan 1st 9999 in *to_date* attribute being a sign of the entry being valid currently.

Name	Type	Size	NULL	Primary key	Index	Integrity constraints
emp_no	INT	4B	N	Y	Y	
title	VARCHAR	50B	N	Y	Y	
from_date	DATETIME	8B	N	Y	Y	
to_date	DATETIME	8B	Y	N	N	

*Table 8: Structure of table **titles***

Name	Type	Size	NULL	Primary key	Index	Integrity constraints
emp_no	INT	4B	N	Y	Y	
salary	INT	4B	N	N	N	
from_date	DATETIME	8B	N	Y	Y	
to_date	DATETIME	8B	N	N	N	

*Table 9: Structure of table **salaries***

After unpacking, the database was loaded from the shell. The following commands were executed:

```
# tar -xjf employees_db-full-1.0.6.tar.bz2
# mysql -p -t < employees_db/employees.sql
```

There was one change introduced to the database after creating as stated above – the *emp_no* column in table *employees*, which is its primary key, was set to auto increment. Following command was run in MySQL prompt:

```
mysql> USE employees
mysql> ALTER TABLE employees MODIFY emp_no INT NOT NULL
    -> AUTO_INCREMENT;
```

In order to provide secure access to data, an user account was created providing only access to data in *employees* database and allowing only select, insert, update, delete and execute operations (execute privilege allows executing stored procedures). User was named *adam* and for the sake of this scenario, a trivial password was used – *abcd1234*. Following commands were run in MySQL prompt:

```
mysql> CREATE USER 'adam'@'localhost' IDENTIFIED BY
    -> 'abcd1234';
mysql> GRANT SELECT, INSERT, UPDATE, DELETE, EXECUTE ON
    -> employees.* TO 'adam'@'localhost';
```

4.1.1 Database migration from MySQL to MS SQL

The original database scripts were written in MySQL format. Since the comparison was needed to be held on the same set of data, it had to be ported to MS SQL database. On a small set of data, such process would be easily carried out by exporting the data into structured text file, but with the database of this size, the size of the exported text file for most tables was too big. And as the MS SQL Server Import and Export Wizard loads all the data from such file into memory upon opening, the system would ran out of memory and did not complete the operation.

Since both testing environments were installed on the same computer, it was impossible

to transfer the data directly. The client computer was used as a mediator – MySQL server version 5.1.45 was installed on it, database *employees* and user *adam* with permission to log from the server's IP address was created. Then all privileges to database *employees* were granted to user *adam*. It was done by running following MySQL commands:

```
mysql> CREATE DATABASE employees;
mysql> CREATE USER 'adam'@'140.134.23.133' IDENTIFIED BY
    -> 'abcd1234';
mysql> CREATE USER 'adam'@'localhost' IDENTIFIED BY
    -> 'abcd1234';
mysql> GRANT ALL ON employees.* TO 'adam'@'140.134.23.133';
mysql> GRANT ALL ON employees.* TO 'adam'@'localhost';
```

The data itself were transferred from server by executing on it:

```
# mysqldump -p employees | mysql -h 140.134.23.139 \
> --user=adam --password=abcd1234 employees
```

Note: transferring the database from server to client could be alternatively substituted with downloading and running the database scripts on client just as on server.

From mediating MySQL database to Microsoft SQL Server on the server, the data were transferred by use of SQL Server Import and Export Wizard. First though, the MySQL Connector/Net data provider [10] had to be installed on the server in order to be able to access the MySQL database. The latest stable version was installed – 6.2.3, which supports the installed version 5.1 of MySQL.

In the SQL Server Import and Export Wizard, the *.Net Framework Data Provider for MySQL* was chosen as a data source, and configured with client's IP address, database name, user name (“adam”) and its password. The SQL Server Native Client 10.0 on localhost was chosen as a destination, with the new database *employees* being created. All the other options were left default.

For each table the specific query was called, selecting all the rows in the table. For example, in case of table *departments*, following query was used:

```
SELECT * FROM departments;
```

The destination table was always renamed according to the source table and each SQL statement creating the table was altered as well, specifying primary key and possibly some other constraints, and changing the column types where they were estimated wrong (for complete table creating scripts, see Annex I.). In case of *employees* table, the *enable identity insert* option was enabled in order to preserver the *emp_no* values as in source database.

Finally the relationships between the tables were established – foreign keys were introduced by running following script as a query in the Microsoft SQL Server Management Studio:

```
ALTER TABLE [dbo].[salaries] WITH CHECK ADD CONSTRAINT
[FK_salaries_employees] FOREIGN KEY([emp_no]) REFERENCES
[dbo].[employees] ([emp_no]) ON DELETE CASCADE;

ALTER TABLE [dbo].[titles] WITH CHECK ADD CONSTRAINT
[FK_titles_employees] FOREIGN KEY([emp_no]) REFERENCES [dbo].
[employees] ([emp_no]) ON DELETE CASCADE;

ALTER TABLE [dbo].[dept_manager] WITH CHECK ADD CONSTRAINT
[FK_dept_manager_departments] FOREIGN KEY([dept_no])
REFERENCES [dbo].[departments] ([dept_no]) ON DELETE CASCADE;

ALTER TABLE [dbo].[dept_manager] WITH CHECK ADD CONSTRAINT
[FK_dept_manager_employees] FOREIGN KEY([emp_no]) REFERENCES
[dbo].[employees] ([emp_no]) ON DELETE CASCADE;

ALTER TABLE [dbo].[dept_emp] WITH CHECK ADD CONSTRAINT
[FK_dept_emp_departments] FOREIGN KEY([dept_no]) REFERENCES
[dbo].[departments] ([dept_no]) ON DELETE CASCADE;

ALTER TABLE [dbo].[dept_emp] WITH CHECK ADD CONSTRAINT
[FK_dept_emp_employees] FOREIGN KEY([emp_no]) REFERENCES
[dbo].[employees] ([emp_no]) ON DELETE CASCADE;
```

Created tables and keys fully correspond with the table definitions in chapter 4.1. There is a slight difference in the column definition that occurred during exporting – the change of type ENUM('M', 'F') to CHAR(1). Microsoft SQL Server does not support ENUM type and since the values listed in it were of CHAR(1) type, it was used instead. To preserve the integrity constraint, a new constraint was introduced automatically during the conversion to the exported database, ensuring that the values in the column will only be 'M' or 'F' (see Annex I.).

In order to provide safe access to the database (allowing only reading and writing, not changing the database structure), new user with login “adam” was created. Created account used SQL Server authentication and for the sake of this scenario a simple password was used (“abcd1234” – the password policy was chosen not to be enforced). User's default database was set to *employees* and user was mapped as a member of roles *db_datareader* and *db_datawriter* for the same database.

4.1.2 Stored procedures in MS SQL

Due to use of custom paging method in GridView ASP.NET component (see chapter 4.2.3.1), the two stored procedures were developed, that this paging method uses. Both procedures use the technique of composing customized SQL query based on a set of conditions and then running the composed query. In this chapter the overview of the procedures is provided, for the whole code see Appendix.

First stored procedure – *GetEmployeesRowCount*, returns the total number of rows in

employees table or its subset based on the search parameters (see Table 10 for the description). It consists of a simple SQL code composing the query returning the row count of the table and attaching the filtering parameters if provided.

	Name	**Type**
Parameters	searchName	VARCHAR(14)
	searchSurname	VARCHAR(16)
	searchDept	CHAR(4)
Returns		INT

Table 10: GetEmployeesRowCount stored procedure description

If all parameters are provided (name: 'abc'; surname: 'def'; department number: 'd001'), the final SQL query run by the *GetEmployeesRowCount* procedure looks as follows:

```
SELECT COUNT(*) FROM employees INNER JOIN dept_emp ON
employees.emp_no = dept_emp.emp_no
WHERE (dept_emp.dept_no = 'd001') AND
(employees.first_name = 'abc') AND
(employees.last_name = 'def')
```

The second stored procedure, named *GetEmployeesPaged*, is returning a subset of the data queried from the *employees* table, to be displayed on one page. Its description is in Table 11.

	Name	**Type**
Parameters	startRowIndex	INT
	maximumRows	INT
	sortExpression	VARCHAR(20)
	searchName	VARCHAR(14)
	searchSurname	VARCHAR(16)
	searchDept	CHAR(4)
Returns		Subset of *employees* table

Table 11: GetEmployeesPaged stored procedure description

The procedure composes a complex query consisting of an inner query, that assigns the row numbers to the table *employees* possibly filtered by name, surname and/or department number, and sorted by the column name passed in parameter *sortExpression*, and outer query. The inner query for *sortExpression* 'last_name DESC', with the search parameters specified as in

the previous example (name: 'abc'; surname: 'def'; department number: 'd001') is following:

```
SELECT employees.emp_no, birth_date, first_name, last_name,
gender, hire_date,
ROW_NUMBER()OVER (ORDER BY employees.last_name DESC)AS row_no
FROM employees INNER JOIN dept_emp ON
employees.emp_no = dept_emp.emp_no
WHERE (dept_emp.dept_no = 'd001') AND
(employees.first_name = 'abc') AND
(employees.last_name = 'def')
```

Outer query is responsible for returning only subset inner query returned data, based on numeric parameters *startRowIndex* (number of a first row to be displayed) and *maximumRows* (number of rows to be displayed). For *startRowIndex* being 40 and *maximumRows* being 10, the outer query is:

```
SELECT emp_no, birth_date, first_name, last_name, gender,
hire_date FROM ( INNER QUERY ) AS numbered_employees
WHERE row_no BETWEEN 40 + 1 AND 40 + 10
```

The right to execute both created stored procedures was granted to the user *adam*.

4.1.3 Stored procedures in MySQL

Since the method of defining the stored procedures differs in both database systems, the scripts creating stored procedures described in the previous chapter had to be rewritten. Also, the predefined function syntax is different and so is the syntax of the composed SQL query itself. The general idea of both procedures – composing the customized query on the basis of few conditions and then running it, and using inner and outer query in case of *GetEmployeesPaged* procedure, are the same, therefore this chapter only focuses on the syntax differences. For the definition of the procedures, see Annex II.

In MySQL the delimiter, which by default is semicolon, is ending the command. Defining a stored procedure is also a command, yet it needs to provide the commands within the procedure ended with a delimiter. In order not to end the whole procedure creating command with the delimiter meant to close the command defined inside the procedure, the delimiter character can be changed. In this way, the delimiter is set to another character (or a sequence of characters) before the create procedure command, which allows to specify the commands inside the procedure with the semicolons. The newly set delimiting sequence is used to close the create procedure command and then the delimiter is changed back to semicolon. The syntax is illustrated in following code:

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE ProcedureName (IN parameter int)
    -> BEGIN
```

```
    -> ...
    -> END//
mysql> DELIMITER ;
```

There is no need to declare variables inside the stored procedure. In MySQL, user-defined variables can are defined simply by assigning them value using *SET* statement. They are however connection-specific, and so the same variable can be accessed inside different stored procedures called in the same connection, hence the variables of the same name used in both defined procedures were renamed, in order to prevent overwriting.

The way of numbering rows in MySQL is based on the user-defined variables. There is no function such as *ROW_NUMBER()* in Microsoft SQL. The rows are numbered by adding a self incrementing statement in the *SELECT* command. The variable is set to 0 before the *SELECT* statement.

Joining the string values in MySQL can not be done by simply adding the two strings – there is a function CONCAT(), that joins any number of strings in specified order. It can however work also with other data types, such as integers, that does not have to be converted to string beforehand.

The execution of composed SQL query also differs – in Microsoft SQL composed string was simply run by *sp_executesql* stored procedure, whereas in MySQL the statement first has to be created from the SQL string, then executed and lastly, the statement has to be deallocated. Following statements allow to do that: *PREPARE*, *EXECUTE* and *DEALLOCATE PREPARE*. The syntax of those is described in following example (assuming that the SQL query is provided in *@sqlcmd* variable):

```
PREPARE statementName FROM @sqlcmd;
EXECUTE statementName;
DEALLOCATE PREPARE statementName;
```

4.2 Web application design

In the case of this scenario there were no demands regarding functionality specified – the goal of the information system was to be implemented with possibly large amount of different components, so that the wide range of ASP.NET code will be run during testing and thus the two implementations ASP.NET implementations will be compared thoroughly.

The structure of information system was based on the database. Development of the web application was carried out in Microsoft Visual Web Developer 2008 Express Edition. Following sub-chapters are describing the different layers of the application.

4.2.1 Data access layer

Data access consists of connecting to the database (or other data source) via data provider and issuing the commands to get or manipulate the data. The first step – connecting to data source can be excluded from the presentation layer by creating the connection string in the *Web.config* file. The data access layer is isolating the second step – the data access logic from the presentation layer, and is responsible for providing the data in the strongly typed objects. Such modular approach is preferred as the potential change of the underlying database system will not affect whole application, but only a single layer.

In terms of ASP.NET, if using a relational database management system, there are two ways to connect to it – using *SqlDataSource* control, or *ObjectDataSource* control. First option requires to configure the control with the explicit SQL statements ant therefore is not distinguishing the data and presentation layer. *ObjectDataSource* on the other hand, can be configured to communicate through the *TableAdapter* defined in the data access layer.

Data access layer (DAL) consists of 4 different kinds of objects. *DataTable* is representing the scheme of a specific view on the database. It can contain whole single table, just a subset of its columns, or joined columns from different tables. It is a class, which instance represents the data table in memory. *Relations* correspond with the relations as defined in the

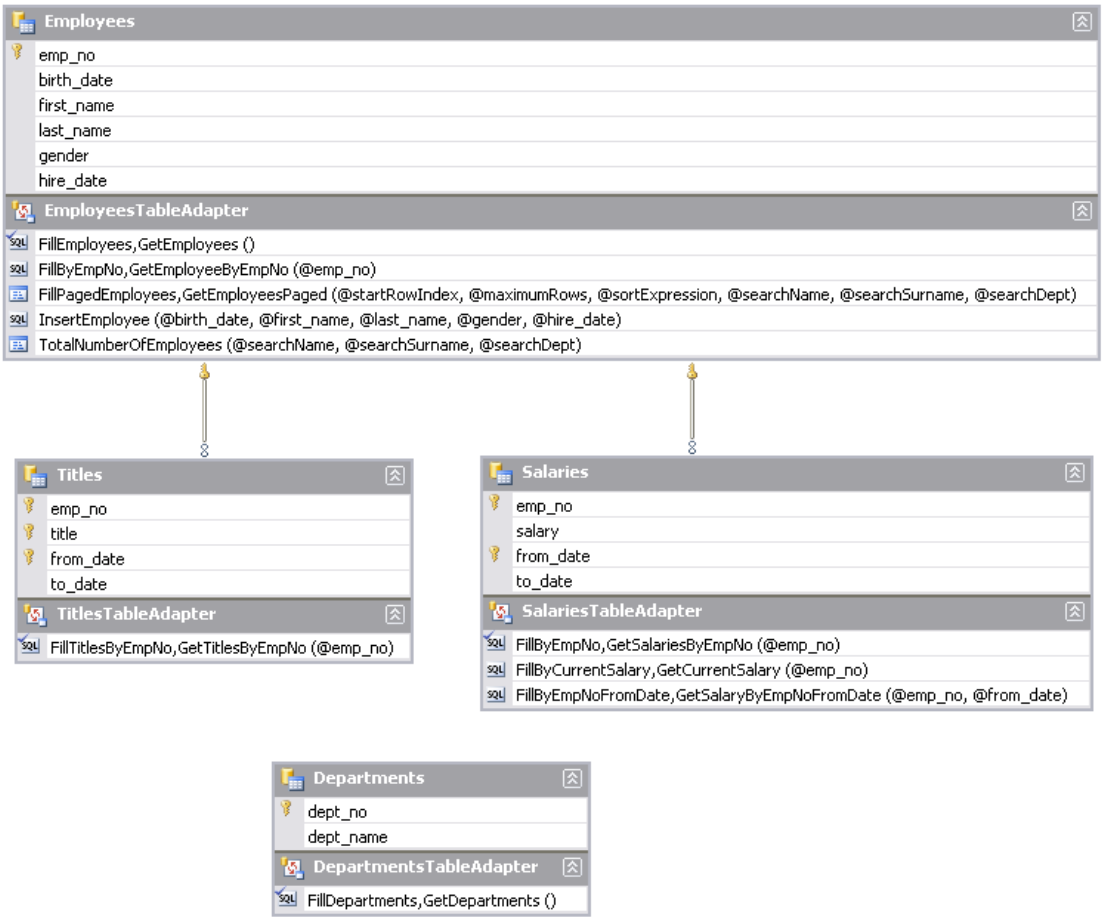


Illustration 2: DAL scheme

database, that data access layer is created to work with. *TableAdapter* is defined for every *DataTable* and is responsible for communication with the database itself. It is issuing the queries or stored procedures and populates the *DataTable* with returned data. Each *TableAdapter* has one or more *Queries*. *Query* in this context is a method of *TableAdapter*, which upon calling issues an SQL query or a stored procedure.

Data access layer is organized into *DataSets* with each database (or database connection) having its own. Since only one database is used in this scenario, single *DataSet* was created – it was named Employees as the database it is representing and saved in subdirectory *App_Code\DAL* in file *Employees.xsd*.

In order to provide wider testing range, both approaches – the one separating data access and presentation layers, and the one that does not, were mixed. Therefore the data access layer was only created for partially. The rest of data were retrieved and manipulated directly from presentation layer via *SqlDataSource* SQL queries. The scheme of created data access layer is to be found on Illustration 2.

All created *DataTables* correspond with the actual database tables. Following paragraph is briefly describing the Q*ueries* in *EmployeesTableAdapter*.

Default select query in *EmployeesDataAdapter* is not used by the above levels. It was used to populate the list of employees before the custom paging method was introduced. Also the default insert method is not used – *InsertEmployee* query is used instead. This query is returning the *emp_no* of newly inserted employee (its execute mode had to be changed from "Non-query" to "Scalar"). Two stored procedures described in chapter 4.1.2 are called via as *TotalNumberOfEmployees* and *GetEmployeesPaged*. Finally, the *GetEmployeeByEmpNo* returns a single row from the *employees* table based on given key parameter *emp_no*.

4.2.2 Business logic layer

Business logic layer (BLL) is an intermediary between DAL and presentation layer. It is used to enforce some policies, that are data-related, but can not be handled by the SQL commands (for example allowing specific operations only on the rows matching some criteria, or changing behaviour of some commands based on such criteria).

In this scenario, the use of business logic layer was inevitable, as the *TableAdapters* from DAL can can be used as the business objects for the *ObjectDataSource* directly, with their *Queries* as the SELECT, UPDATE, INSERT and DELETE methods. However, in order to keep the code relatively complex for the testing, all the *DataTables* from the DAL had had created business layer logic classes for. The classes are put in the *App_Code\BLL* subdirectory.

Selecting methods of BLL classes are in general returning the corresponding *DataTable* returned by appropriate method of respective *TableAdapter*. Other methods are usually just

returning boolean value signalizing whether the operation was successful. The *AddEmployee* and *TotalNumberOfEmployees* methods of *EmployeesBLL* class are returning integer value and BLL is used to convert the object returned by the methods from DAL to integer.

Due to the design of table *salaries*, it needs some extra care to keep the data consistent. For example, in order to set new salary for an employee, it is not sufficient to add an extra row to the table – the currently valid salary is indicated by the value of Jan 1st 9999 in column *to_date*. With adding new entry to the *salaries* table, the last valid entry's *to_date* has to be set to the new entry's *from_date*. That could be done with a stored procedure as well, but was chosen to be done in BLL, in order to make more use of it and rather execute ASP.NET code, than SQL code (as this is the main concern of the scenario). Similarly, upon deletion of the current salary, the previously valid salary is made valid again.

The *SalariesBLL* class is also using batch update access when modifying database data, rather than direct database access used by other BLL classes. The difference between this two approaches is, that while direct database access issues an INSERT, DELETE or UPDATE command immediately for every single change done to the database, whereas the batch update approach works with the *DataTable* in the memory and updates the whole set of data with a single call of Update method on the *TableAdapter* with the *DataTable* as a parameter.

4.2.3 Presentation layer

Presentation layer is responsible for the rendering the content of the page and displaying requested data. It consists of the actual pages, that are, in case of this scenario, split into the design file (aspx) and the code file (aspx.cs). Following aspx pages were implemented:

- Default.aspx
- Departments.aspx
- Employee.aspx
- EmployeeList.aspx
- NewEmployee.aspx

Each of the ASPX pages is using a master page (MasterPage.master), that contains the overhead an the left side navigation menu divided using *div* tags. The menu on the left side consists of *LinkButton* components, that are sending requests using JavaScript functions. The master page also contains CSS code defining the style of the web application. Code file of the master page does not contain any code.

Following sub-chapters are describing individual pages, rendering the content into the structure defined by the master page (with the exception of Default page, that only consists of welcome message and description text, and its code file is empty).

4.2.3.1 EmployeeList

This page is divided into left and right part by use of *div* tags. Left part contains the list of employees and the fields providing filtering options, whereas right part is empty by default, to display the details of the employee selected in the list.

List of employees is rendered by a *GridView* component connected to the *ObjectDataSource.* This couple of components provides a custom paged functionality, meaning, that only the records displayed on currently selected page are returned from the database for every request. This solution uses two stored procedures described in chapter 4.1.2, one of which (*GetEmployeesPaged*) is called by the *ObjectDataSource* through the *EmployeesBLL*'s method *GetEmployees* as the actual select method, and the other, providing the data source component with the number of all the records to be displayed, is assigned to its *SelectCountMethod* parameter via *TotalNumberOfEmployees* method in *EmployeesBLL*. The *GridView* allows sorting by each column. It is realized by passing its *sortExpression* as a parameter by the *ObjectDataSource*.

The filtering (searching) options are provided via two *TextBox* components for entering the name and surname of an employee to search for, and the *DropDownList* component with the list of departments to choose from, which is filled by another *ObjectDataSource*, retrieving data by use of *GetDepartments* method from *DepartmentsBLL*. These components, along with two *Button*s are organized in the *Table* (not the HTML tag, but the ASP.NET component). Both *TextBox* inputs are verified by *RegularExpressionValidator*s, with only small and big letters and input of certain length allowed. The current search options are displayed in the set of *Label*s below the *Table* component.

Right side of the page contains *DetailsView* connected to its own *ObjectDataSource*, that operates on *EmployeesBLL*. It displays the all column values for the record selected in the *GridView* on the left. It also supports editing and deleting of selected record. Each its row, representing one column of *employees* table, has *ItemTemplate* and *EditItemTemplate* (except the one displaying employee number, that only has *ItemTemplate*, as it is not editable). *ItemTemplate* displays data with a *Label* component, that is binded to particular column, whereas *EditItemTemplate* consists of *TextBox* (or *DropDownList* in case of *gender* column) and *RequiredFieldValidator* controls, ensuring that all inputs are presented, and for the date inputs also *RegularExpressionValidator* that assures the date is formatted correctly. Finally, below the *DetailsView*, the *LinkButton* is placed, that allows redirecting to the Employee.aspx page, with the parameter of selected employee number in the URL.

4.2.3.2 Employee

This page displays all the data related to the selected employee. It can only be accessed from the EmployeeList.aspx page by clicking “Detailed employee view” link below selected

employee, that passes selected employee's number as an URL parameter.

In the head of the page, the *FormView* control displays the basic employee data from *employee* table. As it does not provide editing capability, it only has *ItemTemplate* defined – it consists of *Labels* binded to the particular column names organized by some HTML code. This *FormView* is provided with data by *ObjectDataSource* operating with *EmployeeBLL*'s *GetEmployeeByNo* select method.

Follows another *FormView* displaying the department that the employee is currently working in. It allows on viewing and inserting, and thus has *ItemTemplate* and *InsertItemTemplate* defined, first containing *Label* controls binded to the column names of *departments* and *dept_emp* tables, latter composed of *TextBox*es, validating controls and *DropDownList* filled by its own *ObjectDataSource*. The *FormView_department* is provided with data by *SqlDataSource* control. This control bypasses the BLL and DAL layers and connects directly to database executing explicitly stated SQL commands with the parameters being assigned to the data source and commands being run in the assisting code.

In the *ItemTemplate* of aforementioned *FormView* a *LinkButton* “Show history” is placed. This switches from the view of only currently valid department assignment of an employee, to the list of all such previous assignments. This is displayed in by default hidden *DataList* control. The code issued by the *LinkButton* simply hides the *FormView_department* control and displays *DataList_deptHistory* control. Contrary to *FormView*, *DataList* is capable of displaying a list of records. Apart of header and footer templates of the control, the *ItemTemplate* and *EditItemTemplate* are defined in similarly as in a previously described components – displaying of data is handled by binded *Label* controls and their editing by the *TextBox*es verified by *RequiredFieldValidator* and *RegularExpressionValidator* controls. In the *FooterTemplate*, the button to switch back to the default view, showing only current department instead of all the history, is placed. Contrary to the way the button clicked actions were handled in the *FormView* controls, where each button had special method assigned to it, in case of *DataList* the event raising control is the whole *DataList* and the *Button* (or *LinkButton*) is only sending command name in the event. The *DataList_deptHistory* is accessing database via its own data source – *SqlDataSource_deptHistory*.

Salary information are displayed in a way similar to employee's department assignments. In the default view only the currently valid salary is displayed in the *FormView_salary*. Its structure is practically identical to the *FormView_departments*, but it operates on data via *ObjectDataSource* control.

The history of salaries is displayed after clicking on the “Show history” *LinkButton* in the *FormView_salary*, that hides that *FormView* at the same time showing *Repeater1* control, with the list of all past salaries. Similarly to the *DataList* control, *Repeater* is able to display lists of data, but allows more control over the way of displaying it. In order to show data in the

table, the HTML tag starting the table must be placed in the *HeaderTemplate* and tag finishing the table in the *FooterTemplate*. Each *ItemTemplate* (or *AlternatingItemTemplate*, which in case of defining is defining the structure of every second displayed record), must be defined as a table row. *Repeater* also contrary to *DataList* does not support templates for editing or inserting and thus the components for displaying as well as editing have to be placed into *ItemTemplate* and their use is controlled by displaying or hiding them according to user actions. *Repeater1* was sharing its data source with *FormView_salary* – on changing the view, also the select method of the *ObjectDataSource_salary* is changed, so it will select either the currently valid salary, or all of them.

Following is another *Repeater* control, that displays the list of titles, that displayed employee has ever had. *Repeater_Titles*, similarly to the previous one, renders the data with use of HTML table tags. And it also uses the *AlternatingItemTemplate* in addition to *ItemTemplate* to display every second row on a grey background. Editing and deletion are supported by hiding and displaying proper controls. Validating controls are provided for all the inputs. Inserting title records is allowed by the hidden *Panel* control below. In the *FooterTemplate* of the *Repeater* a *LinkButton* "Add Title" is placed, that shows the *Panel_new* control, containing the controls allowing validated input of the new record. *Repeater_Titles* is working with the *ObjectDataSource* connecting to database through *TitlesBLL*, whereas the record inserting button of the *Panel_new* is calling the insert method of the *TitleBLL* instance directly.

Finally there is a *DataList_deptMgr* control, displaying employee's managing responsibilities – that is list of departments that specific user is or was manager of. It has *ItemTemplate* defined with *Label*s binded to the column names and *LinkButton*s with *CommandName* property set, so that clicking on them is handled in the scope of the whole *DataList* (*ItemCommand*). *EditItemTemplate* is defined in a similar way, with editable attributes being displayed in *TextBox*es and checked by *RequiredFieldValidator*s and *ReguralExpressionValidator*s. Below *LinkButton* "Add new" is showing *Panel_deptMgr*, that allows adding of the managing responsibilities. It contains a *DropDownList* allowing to choose the department (it draws data from *ObjectDataSource_deptList*) and *TextBox* for inserting the date, along with the validating controls and *LinkButtons*. Both *DataList_deptMgr* and *Panel_deptMgr* are working with *SqlDataSource_deptMgr*.

The page uses its *Page_Load* event to supply all the data sources that work with the employee number with its proper value when it is first loaded. The employee number is passed as a parameter in URL.

Because the database uses date of Jan 1st 9999 as a symbol for currently valid records and displaying it as such would be confusing and unsightly, every control displaying *to_date* column of any table is handling its *PreRender* event in order to change the string containing such date to word "current" or "now". Similarly, the input controls bound to *to_date* columns

are checked for entries “current”, “now” or empty string, and if such is found it is being treated as the date of Jan 1st 9999.

In order to facilitate the process of creating new employee's record, the *FormView* controls displaying salary and department assignment are automatically switched to the insert mode if there are no records to display. It is done by handling the *DataBound* events on the *FormView*s.

4.2.3.3 NewEmployee

This page only contains two controls – *DetailsView* and its data source *ObjectDataSource*. The *DetailsView* control has all fields defined manually as *TemplateField*s with *InsertItemTemplate*s, inside which *TextBox* controls validated by *RequiredFieldValidator* and *RegularExpressionValidator* controls are placed. Exception is the gender field, where *DropDownList* allowing to choose only from “M” and “F”, not requiring validation. *ObjectDataSource* only issues insert command, calling *AddEmployee* method from *EmployeesBLL*.

As this page only serves for adding the employees, the *DetailsView* is switched to the insert mode upon loading of the page (in *Page_Load* event). After inserting of the employee, page automatically redirects to Employee.aspx with returned new employee number as URL parameter. This is handled as *ObjectDataSource*'s *Inserted* event. In case of cancelling the inserting dialogue, the *DetailView*'s *ModeChanged* event handles redirecting page to *Defalut.aspx*.

4.2.3.4 Departments

Page displaying list of departments is divided into left and right part by use of HTML *div* tags. On the left the departments themselves are shown, whereas in the right part list of current and former managers is displayed upon selecting specific department.

On the top of the right part the *GridView* is showing the list of departments. It allows selecting, editing and deletion of the departments. Its data source is *ObjectDataSource* connecting to database with use of *DepartmentsBLL*.

Below the *GridView* is the *LinkButton* “Add new department” and *FormView* used for adding new department records. The *FormView* only has *InsertItemTemplate* defined, therefore on page load it does not render at all. Only upon clicking at the “Add new department” button, it changes the *FormView*'s mode to insert, it renders the content of *InsertItemTemplate*. This template consists of TextBoxes for entering new departments name and number validated by RegularExpressionValidators and RequiredFieldValidators, and LinkButtons with CommandName properties specified, so that their activation can be handled automatically by

the FormView. All the controls are organized into HTML table. *FormView* is sharing the same *ObjectDataSource* with above *GridView.*

The right side of the page displays managers of selected department in *DataList*. By use of HTML tags in *ItemTemplate* it renders a link with the name and surname of the manager redirecting to the Employee.aspx page with that manager's employee number as an URL parameter. It further renders the dates between which the current employee was managing the department.

As in case of Employee.aspx page, also here dates are shown containing Jan 1st 9999 as a value symbolizing current validity of the entry. Therefore *DataList*'s event *PreRender* ichchecking for those values and changing them into word "now" to improve the aesthetics.

4.3 Porting web application to Mono

This sub-chapter is dedicated to the description of all changes necessary to introduce to the application created in Microsoft Visual Web Developer and run on Windows Web Server 2008. It is divided to into subsections according to the sections according to the sub-chapters of Web application design chapter.

4.3.1 ADO.NET

For connections to the Microsoft SQL database a standard *SqlClient* driver was used. However, in order to be able to connect to the MySQL database, an alternative ADO.NET driver has to be used. There is a possibility to use an ODBC driver, but this adds an extra layer and thus is likely to provide worse performance, therefore the Connector/Net driver developed by MySQL AB, just as MySQL database system itself.

The Connector/Net driver was obtained from the MySQL developer web pages in version 6.2.3.0. It consists of a group of DLL files, of which only one – *mysql.data.dll*, was used and classes provided in *System.Data* library were substituted by the classes from *MySql.Data* library. The *mysql.data.dll* file was placed in the *bin* directory in the web application's root directory and reference to it was added to the Web.Config file as follows:

```
<configuration>
  <system.web>
    <compilation>
      <assemblies>
        <add assembly="mysql.data, Version=6.2.3.0,
          Culture=neutral, PublicKeyToken=c5687fc88969c44d"/>
      </assemblies>
    </compilation>
  </system.web>
</configuration>
```

The *PublicKeyToken* for the DLL can be determined by the use of *sn.exe* executable, that is a part of .NET SDK. To get the token for the *mysql.data.dll* following command was run:

```
> sn.exe -T mysql.data.dll
```

To be able to use the *MySqlClient* as data provider, the *DbProviderFactories* section had to be created in the Web.Config and provider for *MySqlClient* added. The Visual Studio in its Express edition does not support any alternative plugins such as MySQL plugin for Visual Studio, that provides automated generation of all the DAL code, nor does it support adding any 3rd party data providers, however one can still define it manually. Visual Studio is then unable to debug the project with such alternative data provider defined, but it does not mention it as an error in the syntax.

Using the Professional edition of Visual Studio would surely be easier, but the Express edition was chosen to show, that it is possible to port the applications developed for the Microsoft platform to Mono using only the free development tools. Furthermore, the inability to uses the MySQL plugin for Visual Studio, leads to the manual editing of the DAL, and thus illustrates the structural differences between *System.Data* library and *MySql.Data* library.

The data provider was defined in Web.Config in a following way:

```
<configuration>
  <system.data>
    <DbProviderFactories>
      <add name="MySQL Data Provider"
        invariant="MySql.Data.MySqlClient"
        description=".Net Framework Data Provider for MySQL"
        type="MySql.Data.MySqlClient.MySqlClientFactory,
          MySql.Data, Version=6.2.3.0, Culture=neutral,
          PublicKeyToken=c5687fc88969c44d" />
    </DbProviderFactories>
  </system.data>
</configuration>
```

Finally the connection string had to be modified. Since the variables such as user name, password and database location were kept the same for both Microsoft SQL database and MySQL database, only the *providerName* attribute had to be altered – it was changed to *MySql.Data.MySqlClient*.

4.3.2 Data access layer

Created DAL is an XML file, that represents configuration of the *DataSet*. Upon running the project in Visual Studio, or on request of the page in IIS, the XML file is used to generate the the code file, that can be compiled. Mono does not provide such automatic conversion and the *DataSet* code file has to be created manually. The tool allowing to do so is a

part of Mono, just as .NET SDK. It is in both cases it is a Win32 executable named *xsd.exe*. In this case the Microsoft's executable was used and the *Employees.cs* file was generated by running following command:

```
> xsd.exe Employees.xsd /d
```

The output file was placed in App_Code directory instead of whole subdirectory DAL. It was done so in both compared environments, so to create equal conditions.

The *DataSet* code file however had to be changed, as is had been generated to use with Microsoft SQL Server. In order to use it with the MySQL through the Connector/Net driver. Since the driver provides different classes and their syntax differs as well, some batch editing had to be done upon the *DataSet* code file. Following table describes the changes introduced (in order as stated).

Original *DataSet* code file	***DataSet* modified for use with MySQL**
`global::System.Data.SqlClient.SqlDataAdapter`	`MySql.Data.MySqlClient.MySqlDataAdapter`
`global::System.Data.SqlClient.SqlConnection`	`MySql.Data.MySqlClient.MySqlConnection`
`global::System.Data.SqlClient.SqlCommand`	`MySql.Data.MySqlClient.MySqlCommand`
`global::System.Data.SqlClient.SqlParameter`	`MySql.Data.MySqlClient.MySqlParameter`
`global::System.Data.SqlDbType.Int`	`MySql.Data.MySqlClient.MySqlDbType.Int32`
`global::System.Data.SqlDbType.Char`	`MySql.Data.MySqlClient.MySqlDbType.VarChar`
`global::System.Data.SqlDbType`	`MySql.Data.MySqlClient.MySqlDbType`

Table 12: Class name differences between Microsoft SQL ADO.NET driver and MySQL Connector/Net

Also the parameters definition of two corresponding classes – *SqlParameter* and *MySqlParameter* differs and so the code had to be adjusted in changing the parameters order and skipping some of it. Illustration 3 describes the corresponding parameters.

The function returning the automatically incremented primary key of the last inserted row has different name in both database systems – therefore the *SCOPE_IDENTITY()* was changed to *LAST_INSERT_ID()*.

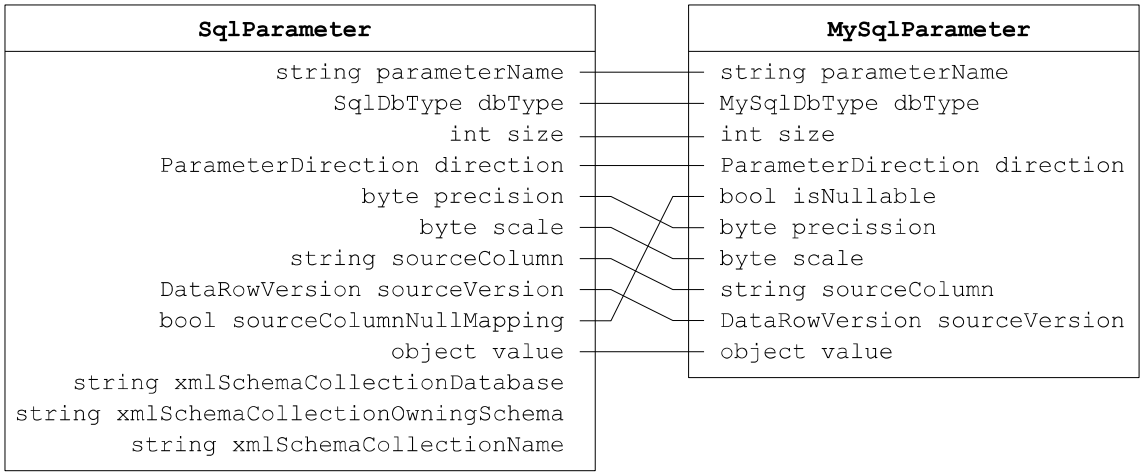


Illustration 3: Constructor parameters comparison for classes SqlParameter and MySqlParameter

Finally, the statements referring to the tables and stored procedures inside the database in Microsoft SQL were generated including the *dbo* schema specified and the names of the database, tables and columns were often enclosed in square brackets. This syntax is not recognized by MySQL and those had to be removed.

4.3.3 Presentation layer

Presentation layer did not require much changes, as most of the database operations were defined in DAL and the, and the .NET classes implementation by Mono should be compatible with Microsoft's one.

However, where the *SqlDataSource* components were used (on pages *Employee.aspx* and *Departments.aspx*), the SQL syntax had to be adjusted as in the case of DAL. Specifically deleting the square brackets around the table and column names had to be done, just as deleting the *dbo* scheme where it was specified.

Besides this database related changes, there was only one change introduced. On the *Employee.aspx* page, there were two *Repeater* controls, both rendering *TextBox*es and their validating controls (*RequiredFieldValidator* and *RegularExpressionValidator*). In Microsoft environment this combination was working properly, but if run under Mono, the *LinkButtons*, that had their *CausesValidation* property set to “true”, did not raise the *PostBack*. Apparently the JavaScript code was not generated properly. This issue has been reported as a bug in Novell's Bugzilla [11]. If however the *LinkButtons* causing validation had their *CausesValidation* property set to “false”, the event was raised properly. Therefore, as a workaround, the validating controls were handled in the event servicing code, by calling their *Validate* method and only

proceeding in event handling, if their *IsValid* property is “true”. This should not reduce performance, if the values provided in the validated controls are correct. However if the validation is automatic, it is carried out in two steps – locally and server-side, whereas if the inputs are validated manually in the event code, there is no local validation provided by the JavaScript Therefore in case, that provided values are not matching the criteria controlled by validators, those values are sent to server anyway, which would be avoided if the validation would be automatic. In other words, if the provided input data are invalid, there is unnecessary *PostBack* raised. The manual validation of controls inside *Repeaters* was used for both testing environments.

5 Testing

The testing of the information system was performed from the client computer and its main goal was to compare the efficiency of hosting the web application in both environments. The functionality of the was tested manually during development and later during porting, therefore it is not mentioned in this chapter, however some functionality testing tools were used for the efficiency testing as well and thereby it was assured, that both solutions are providing the same results. This chapter is divided into sub-chapters according to the tools used for those purposes.

Before the testing began, the debugging mode of the web application had to be turned off. The *Web.Config* files were edited as follows:

```
<configuration>
  <system.web>
    <compilation debug="false" />
  </system.web>
</configuration>
```

All tests were run on an isolated network consisting only of client computer and server computer connected by crossed UTP cable. The network was operating on 1 Gbps speed. This architecture was used to minimize the network's influence on the tests.

The browser used in some of the tests was Mozilla Firefox 3.6.3 with new profile and no extensions (except the latter installation of *Selenium IDE* – see chapter 5.2).

5.1 Page loading times

First efficiency comparison was based on the page loading times. It simply involved manually requesting a page in the browser and waiting for the response. The network traffic was observed on the client's side by the Wireshark open source network packet analyser (latest stable release – version 1.2.9, was used) [12]. The times between sending GET request and receiving OK response were measured.

Page loading times were inspected on 3 different pages – *Default.aspx*, which does not request any data from the database, *EmployeeList.aspx*, which runs relatively complicated queries and uses stored procedures, and *Departments.aspx*, which requests only a small amount of data, with a simple SQL query.

First request of every page was done against freshly booted up server, to see how long does it take the services to respond the first time. Following (second to fifth) requests were served in more or less the same time and therefore an average time was calculated from them to be shown in graph for comparison. Response times are displayed in Table 13 for Linux

environment and Table 14 for Windows server.

		Request					2^{nd} – 5^{th} average
		1^{st}	2^{nd}	3^{rd}	4^{th}	5^{th}	
Page	*Default.aspx*	8,025486	0,007244	0,004733	0,004320	0,004264	0,005140
	EmployeeList.aspx	10,528721	1,719087	1,724751	1,711849	1,714068	1,717439
	Departments.aspx	8,641694	0,024922	0,014047	0,013077	0,013276	0,016331

Table 13: Linux server response times in seconds

		Request					2^{nd} – 5^{th} average
		1^{st}	2^{nd}	3^{rd}	4^{th}	5^{th}	
Page	*Default.aspx*	3,755343	0,004086	0,003988	0,002754	0,004089	0,003729
	EmployeeList.aspx	8,682184	0,110965	0,061883	0,057641	0,057291	0,071945
	Departments.aspx	6,671512	0,022013	0,010790	0,008617	0,007225	0,012161

Table 14: Windows server response times in seconds

The differences are apparent especially on loading the *EmployeeList.aspx* page, the one running the most complicated SQL queries. Linux server is considerably slower in providing response especially on the repetitive requests. In order to determine, what is responsible for such a delay – Mono and it's connector to the database, or the MySQL database itself, the database query execution times were also compared.

Page *Default.aspx* is not running any queries, therefore no measurement was needed for this one. The other two tested pages are running following queries on loading:

EmployeeList.aspx:

```
SELECT * FROM departments;
GetEmployeesRowCount(NULL, NULL, NULL);
GetEmployeesPaged(0, 20, NULL, NULL, NULL, NULL);
```

Departments.aspx:

```
SELECT * FROM departments;
```

Those queries were run in both – MySQL prompt in Linux and SQL Server Management Studio. In order to measure the times of execution different methods had to be used in both environments. MySQL has a built in profiling tool, that allows to determine the execution time in microseconds. It is activated by setting local variable *profiling* to 1 and the execution times for the recent queries can be printed with *show profiles;* command. Before the first execution of the first query of the page, the cache was cleaned with *flush tables;* command,

so that the result would correspond with the first page request after booting up the system. The first run of the queries associated with *EmployeeList.aspx* page was following:

```
mysql> FLUSH TABLES;
mysql> SET profiling=1;
mysql> SELECT * FROM departments;
mysql> CALL GetEmployeesRowCount(NULL, NULL, NULL);
mysql> CALL GetEmployeesPaged(0, 20, NULL, NULL, NULL, NULL);
mysql> SHOW PROFILES;
```

Following executions were lacking the first two statements, as the cache did not need to be cleaned and profiling was already turned on. After executing the last query and checking the profiles, the profiling was switched off by running *set profiling=0;* command.

The *show profiles;* command prints the execution times even for the single commands inside the stored procedure, therefore all the durations have been added up. By far the slowest of all commands was the *EXECUTE* statement inside the *GetEmployeesPaged* stored procedure (see Table 15). By examining the query executed by this statement, it was revealed, that on such large database with InnoDB engine, MySQL has a serious deteriorate problem with executing a queries consisting of inner query.

		Request					**2nd – 5th average**
		1st	**2nd**	**3rd**	**4th**	**5th**	
Associated Page	*Default.aspx*	0,000000	0,000000	0,000000	0,000000	0,000000	0,000000
	EmployeeList.aspx	1,662280	1,664862	1,654331	1,651557	1,660920	1,657918
	Departments.aspx	0,017257	0,000300	0,000283	0,000286	0,000272	0,000285

Table 15: MySQL query execution times in seconds

With Microsoft SQL Server, the execution time of different commands is done by comparing the timestamps from before and after executing the command (or commands). In order for the timestamps accuracy to be in microseconds, the *datetime2* variable was used. The whole sequence of commands used to determine the execution time of queries associated with *EmployeeList.aspx* page was as follows (with first two commands cleaning the cache, therefore used only before the first measurement):

```
DBCC DROPCLEANBUFFERS
DBCC FREEPROCCACHE
DECLARE @startTime datetime2;
SET @startTime = SYSDATETIME();
SELECT * FROM dbo.departments;
EXEC GetEmployeesRowCount NULL, NULL, NULL;
EXEC GetEmployeesPaged 0, 20, NULL, NULL, NULL, NULL;
```

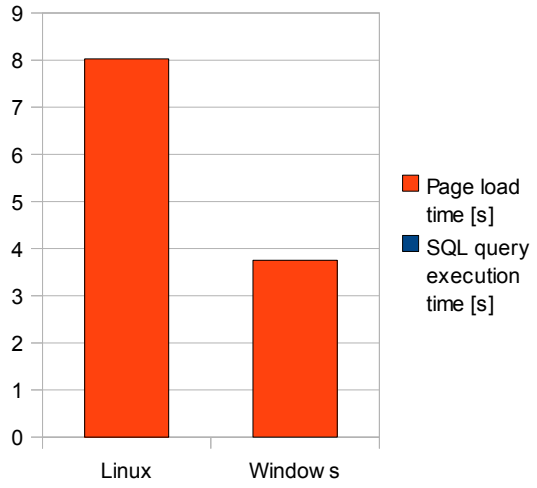
```
SELECT exectime = DATEDIFF(MICROSECOND, @startTime,
      SYSDATETIME());
```

The results are presented in Table 16.

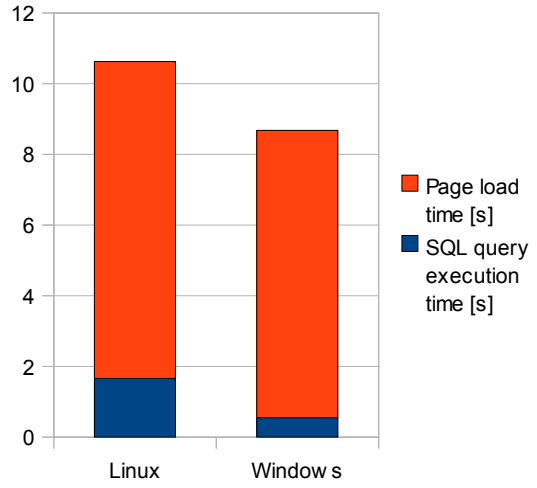
		Request					2nd – 5th average
		1st	2nd	3rd	4th	5th	
Associated Page	*Default.aspx*	0,000000	0,000000	0,000000	0,000000	0,000000	0,000000
	EmployeeList.aspx	0,549769	0,074214	0,068355	0,072261	0,077143	0,072993
	Departments.aspx	0,018553	0,000000	0,000000	0,000000	0,000000	0,000000

Table 16: MS SQL query execution times in seconds

The data from the tables are depicted in the following graphs. Graphs 1 to 3 are showing the duration of page loading and the share of the SQL query execution on it.



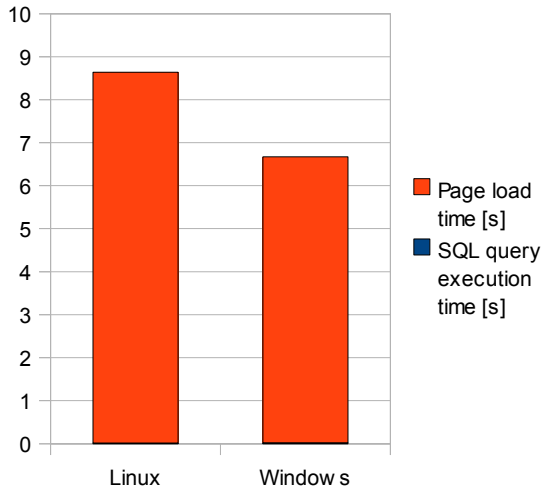
Graph 1: Duration of operations during first loading of Default.aspx page



Graph 2: Duration of operations during first loading of EmployeeList.aspx page

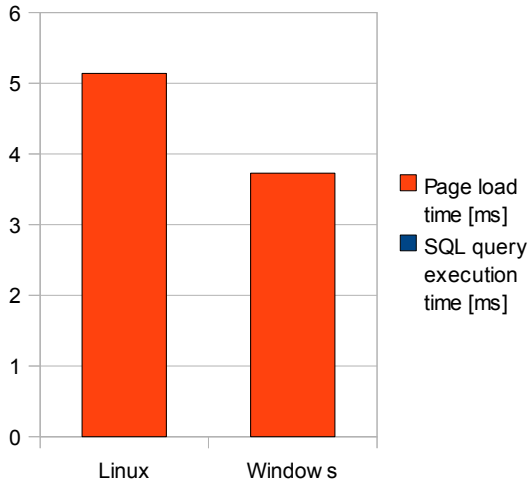
As there are no SQL operations executed upon loading of *Default.aspx* page and the query used in *Departments.aspx* is very simple, the only page significantly slowed down by the database operations is *EmployeeList.aspx*.

Any page, when loaded on freshly booted-up server had the user waiting for the significant time, yet as long as it only occurs first time after the server is started, it presents no performance drawbacks.

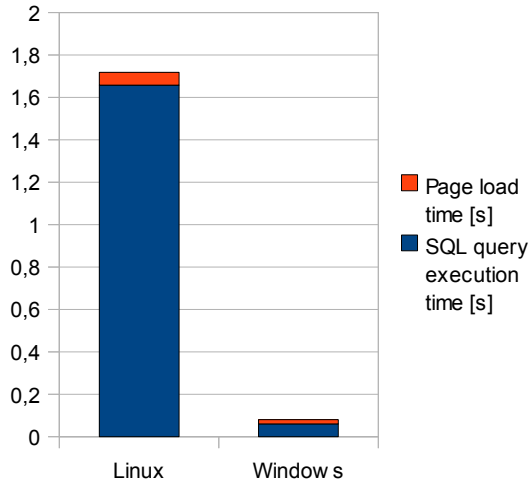


Graph 3: Duration of operations during first loading of Departments.aspx page

Further graphs (4, 5 and 6) are depicting the average durations of page loading and respective database operations for the second and subsequent requests.

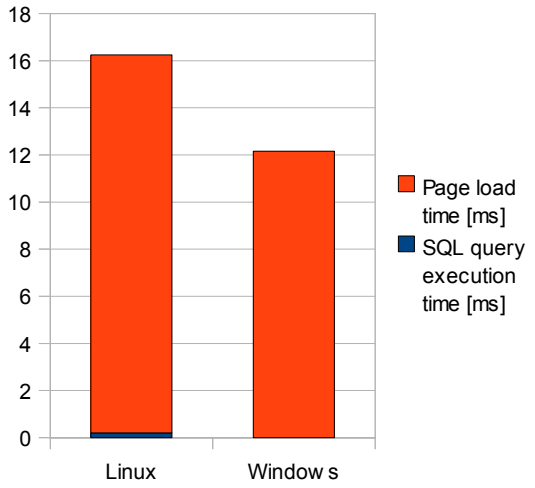


Graph 4: Average duration of operations during loading of Default.aspx page



Graph 5: Average duration of operations during loading of EmployeeList.aspx page

It is clear from the graph that for *EmployeeList.aspx* page the database operations constitute most of the page load time, yet it is significantly more in Linux environment. While graphs 4 and 6 have their scale in milliseconds, graph 5 renders the scale in seconds. Even the manual request of the page against the Linux server of the information system caused a noticeable delay.



Graph 6: Average duration of operations during loading of Departments.aspx page

5.2 Duration of functionality tests

In order to measure the duration of execution of other operations than those occurring during the page loading *Selenium IDE* tool was used. This tool is first of the three *Selenium* tools used for automated functionality testing. It comes as a extension to *Firefox* browser and allows to record all actions that user does through the browser, as well as manual creation of tests and editing of the recorded test sequences. It also allows running the tests from the *Firefox* browser it is installed in. Another two *Selenium* tools are *Remote Control* and *Grid* – first allows remotely run the tests in different browsers, whereas the latter can be used to synchronize few *Selenium Remote Control* instances to simultaneously run the tests, which is most commonly used to run the tests on different platforms at the same time [13]. All parts of *Selenium* system are licensed under Apache 2.0 license.

The tests designed in *Selenium IDE* were covering all functionalities of the developed information system and were written and sequenced to leave the database in the same status after finishing the whole cycle, as it was in the beginning (with the exception of *AUTO_INCREMENT* or *IDENTITY* status, which was obviously incremented by entering new values, and was left so even after their deletion). In this way, the same test could be re-run multiple times.

The test cases created by *Selenium IDE* are saved as a HTML files and they have a structure of a table with 3 columns – first defines the command, second the target the command is performed upon, and third is a value argument.

Used Selenium commands:

- open(url) – opens the page specified as url target and waits until it loads
- ClickAndWait(locator) – activates the link or button specified in locator target
- type(locator, value) – sets the content of input field specified by the locator target to the *value*
- select(locator, option) – selects the *option* from the drop-down list specified by the *locator*
- *assertTextPresent(text)* – verifies the presence of *text* on the page; if not found, the test fails and stops
- *assertText(locator, text)* – verifies the presence of *text* in the element specified by the *locator*; if not found, the test fails and stops
- assertElementNotPresent(locator) – verifies that the element specified by the locator in not on the page; if found, the test stops and fails

The *locator* parameters were specified by element's *id* on the page or via *XPath*. All the test cases were recorded from the user actions in browser and only the *assert* statements verifying the proper content were added manually. The test cases are organized into test suite, so that they are executed in the right order. Following list presents the order of the test cases and the description of operations run in each one.

List of created test cases:

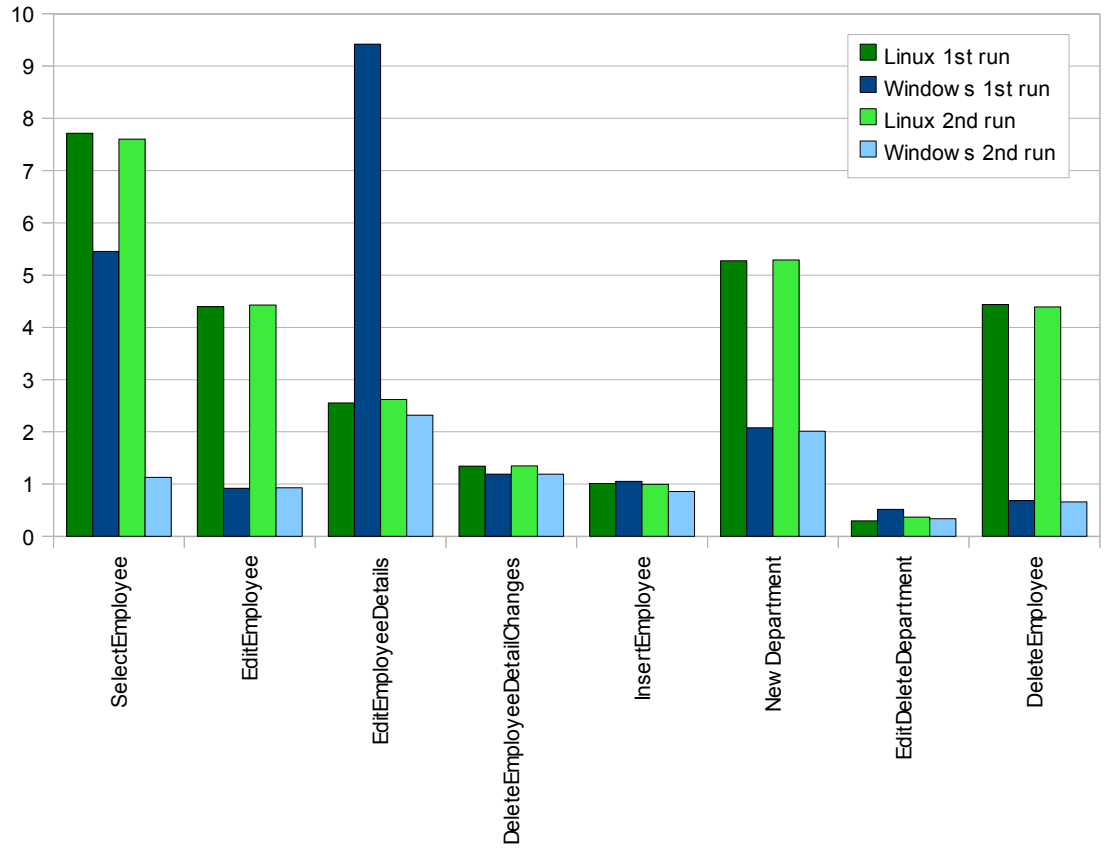
- *SelectEmployee* – searches for employee named 'Woody' working in 'Quality Management', scrolls to page no. 2, sorts list according to surname, selects employee 'Woody Mamelak' and verifies if proper information is rendered.
- *EditEmployee* – selects employee 'Christian Koblick', edits him and changes his name to 'David', date of birth to '1.5.1952', gender to 'F' and updates the information. Then verifies if the changes were done properly, and edits the employee information again, undoing the changes.
- *EditEmployeeDetails* – opens the *Employee.aspx* page of employee 'Danae Heping', sets his assignment to 'Production' department to finish on 12.3.2002 and verifies the change. Then assigns him to 'Finance' department from 12.3.2002 until now, sets his salary to 82 000,- Kč (valid since 4.8.2005) and checks the salary changes. Follows with adding a new title 'Senior Engineer' valid since 12.3.2002 until now and edits the validity of a previous title until 12.3.2002 as well. Finally adds the managing responsibility for the employee on the 'Finance' department since 4.8.2005, then edits it and finishes it to 1.1.2008 and verifies all the changes.
- DeleteEmployeeDetailChanges – revokes all the changes introduced to the employee 'Danae Heping' by the previous test case.

- *InsertEmployee* – inserts a new male employee 'Ralph Morne', born on 13.5.1976 and hired on 24.9.1999, assigns him to the 'Human Resources' department with salary of 49 000,- Kč and title 'Spokesman', all valid from 24.9.1999 on. Then checks all the values.
- *NewDepartment* – creates new department with number 'd010' and name 'Public Relations', then searches for the employee called 'Ralph' selects the one created in the *InsertEmployee* test case, and edits him, so that he is assigned to the newly created department from 1.1.2003 and his previous assignment is ended to the same date, then sets him as a manager of the new department since 1.1.2003 too, and verifies the changed values.
- *EditDeleteDepartment* – changes the name of department created in the previous test case to 'News Department', verifies the change and then deletes the department, and assures that it is gone.
- *DeleteEmployee* – searches for 'Ralph Morne', deletes him and checks if the entry is removed.

The tests cases were run in two series – at first each test case was run separately and the time between the first GET request and the last OK response was measured using Wireshark just as in the previous testing phase. This was repeated twice – first time the system was freshly booted up, but before actual testing the 3 pages were manually requested – *Default.aspx*, *EmployeeList.aspx* and *Departments.aspx*. In this way the post boot up delay was prevented without providing the cache with much data, so that the caching efficiency could still be observed between the first and second run of the test suite. Table 17 presents the results of the first testing phase and Graph 7 visually compares test cases execution times.

		Linux test run		**Windows test run**	
		1st	**2nd**	**1st**	**2nd**
Test case	*SelectEmployee*	7,716137	7,603845	5,455114	1,129443
	EditEmployee	4,398392	4,426866	0,919986	0,930601
	EditEmployeeDetails	2,554092	2,619332	9,420467	2,318799
	DeleteEmployeeDetailChanges	1,342007	1,347402	1,189763	1,186023
	InsertEmployee	1,011877	0,998357	1,054796	0,863044
	NewDepartment	5,272004	5,292445	2,081637	2,009285
	EditDeleteDepartment	0,299922	0,367643	0,516403	0,342581
	DeleteEmployee	4,439008	4,393642	0,687673	0,663166
	SUM	27,033439	27,049532	21,325839	9,442942

Table 17: Test cases separate run times in seconds



Graph 7: Selenium test cases execution time comparison

It is clear from the graph, that while Microsoft platform benefits from caching in some cases, on Linux the execution times practically do not differ. Therefore if in the first run the total time of running all test cases was comparable on both platforms (although the times of running particular test cases differ significantly), on second run Windows is almost 3 times faster.

Following table presents further execution times of whole test suite. Note, that the times of executing the whole suite are approximately 1,5 seconds longer than the sum of 2nd execution times of single test cases. It is so due to the fact, that the tests are operated on the web browser level, which means, that the information does not only have to be received, but also rendered by the browser, which causes some delays between receiving OK response from a web server and sending a new request. Since this occurs also during the execution of single test cases with the same delay effect for both tested environments, the difference between the Linux and Windows servers response times differences are even more significant than it appears on Graph 7.

Environment	3rd execution	4th execution	5th execution	Average
Linux	28,64	28,33	28,50	28,49
Windows	11,26	10,93	10,90	11,03

Table 18: Selenium test suite execution times in seconds

As the *KeepAlive* property of both web servers – Apache and IIS, was turned off (see chapters 3.1.3.2 and 3.2.1), the tests were also run with *KeepAlive* on, but the results were even slightly slower (in both cases approximately 0,5 s on the whole suite run).

5.3 Load testing

Previously conducted tests were simulating a single user only. It is however often more important how many users the server can handle at the same time, than how quickly will it respond to user's request. Load testing is a simulation of multiple clients trying to request server's resources.

In this scenario an open source web performance tool *Pylot* (version 1.26) was used [14] on the client computer. It is written in *Python* and so requires its installation. Along with *Python* (in version 2.6) the libraries *NumPy* and Matplotlib were installed, as *Pylot* uses them to render graphical output. Although *Pylot* also includes GUI, it requires installation of further *Python* extensions and therefore it was used as a command line utility only.

Pylot allows definition of the test cases in the XML file *testcases.xml* placed in its root directory. The structure of the file as defined for this test suite was as follows:

```
<testcases>
  <case>
    <url>http://140.134.23.133</url>
    <method>GET</method>
    <verify>Home page</verify>
  </case>
</testcases>
```

There can be any number of *case* elements declared inside *testcases* element. The only required element inside *case* is *url*. If no other options would be specified, the GET request would be sent to specified URL and the success of the test run would be assessed on the basis of returned HTTP code. In this scenario however, the expected text present on the page was always verified (the *verify* element inside the *case* can also repeat). The designed set of test cases contains one *case* requesting particular web page of the information system, with the exception of *Employee.aspx* page, that is being requested in 5 different *case*s with different *emp_no* passed as parameter. For the precise definition of the test cases please refer to Annex III.

Command line interface of *Pylot* was run as follows:

```
python run.py
```

With following parameters specified:

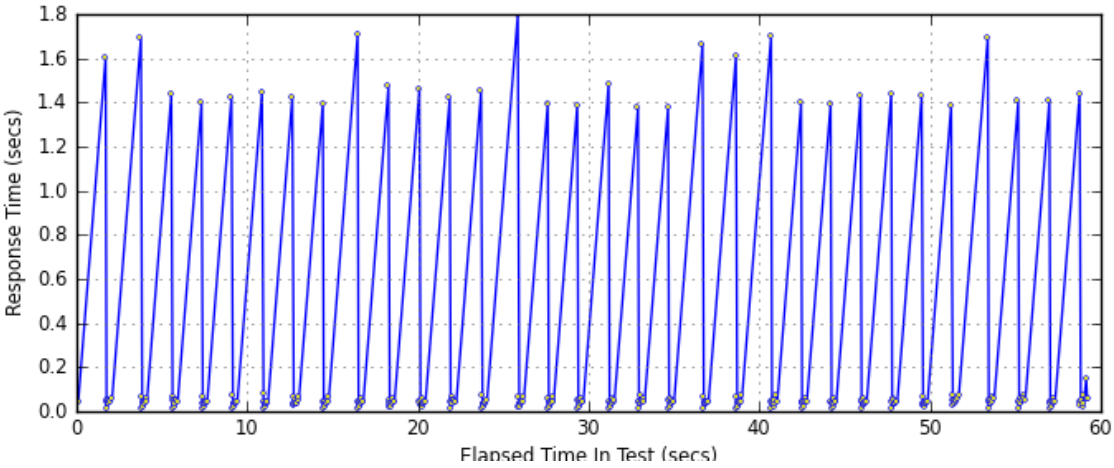
- -a number of agents, virtual users sending requests to the server
- -d duration of testing in seconds
- -i interval of sending requests by agent in milliseconds

By default the interval of sending requests is set to zero, but each agent always waits for a response first before requesting another page. The *config.py* file placed in *core* folder of *Pylot* was modified and *SHUFFLE_TESTCASES* was set to true, so that the agents will run the test cases randomly.

Pylot generates detailed reports of testing in HTML format – all the reports are to be found on attached CD, each in separate folder with folders named according to the following logic (where Environment can either be "Windows" or "Linux" and "[num]" are numbers as specified for particular parameters in command line when running the test):

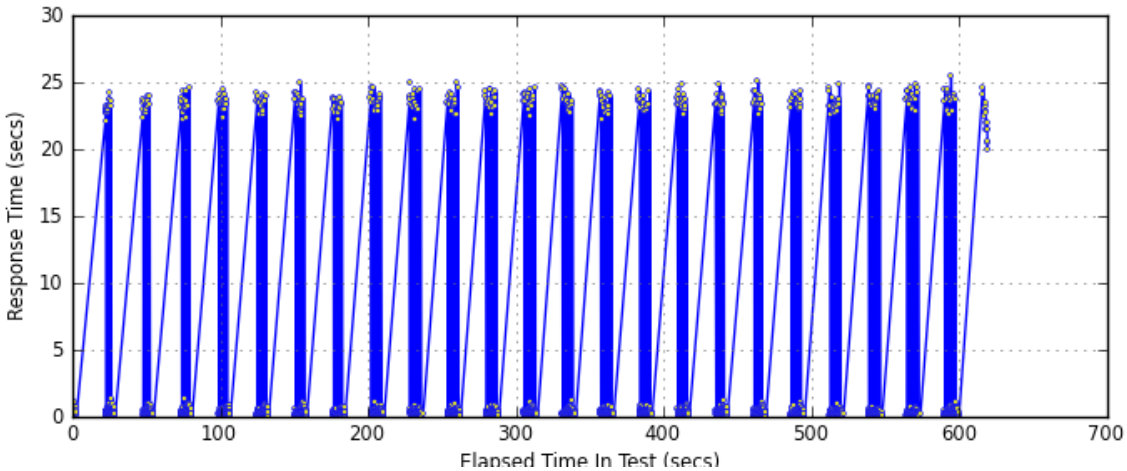
```
Environment_a[num]_d[num]_i[num]
```

First the Linux environment was tested. The test with one agent and zero interval between requests was run for 60 seconds. During that run, MySQL process was taking 40-50% of CPU time, whereas Mono and httpd2-worker processes (Apache web server process) were taking insignificant share (never more than 2%). The Graph 8 shows very unbalanced response times of the server – and knowing, the page load times from the first conducted tests, it is the test case requesting page *EmployeeList.aspx* responsible for the peaks on the graph.



Graph 8: Response times of Linux server for single agent

Gradually rising the number of agents, the maximum clients that server can handle was determined. The maximum number of agents, that server was still able to respond without sending the error messages was 20. It was tested for 10 minutes and no errors occurred, yet the response times for some requests have risen up to 25s (as can be noticed from Graph 9).



Graph 9: Response times of Linux server for 20 agents

During that test the MySQL process was consuming 97% of CPU time most of the time, however during the graphs low extremes, many requests were served at the same time, which made Mono using 26% of CPU time, and MySQL 52% at that time. Apache processes never took more than 2% all together.

When further rising the number of agents, server could not handle the requests and started to send HTML responses 503 – Service Unavailable. With 25 agents there were 22 such error responses returned out of 540 requests in 60 seconds.

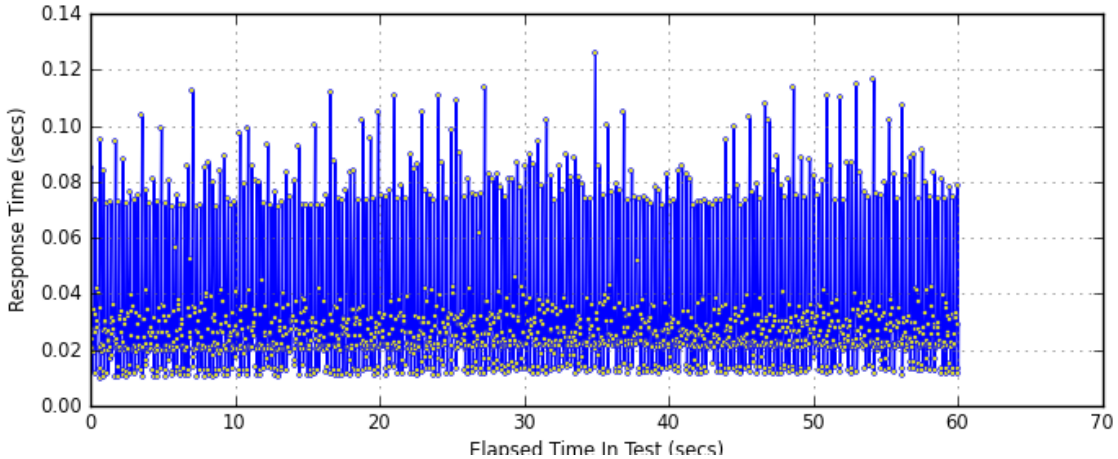
However, constant load from the same users is something that rarely happens on the web server in the normal use. Therefore the *interval* parameter was set to 3000 (milliseconds), which is more like human would behave, browsing the pages. With this settings the server could handle 30 agents without errors. When the requests have been sent by 35 agents simultaneously, the server responded with 503 error 2 times during one minute.

When testing the Windows server the same procedure was followed. First a single agent constantly requesting the pages was run. During the test run the SQL server and *w3wp.exe* (web server process) were each taking about 11% of CPU time on average. The Graph 10 is showing much more balanced responses than in case of Linux server.

The limit above which the Windows server would start to send an error responses was not discovered. When trying to flood the server with 900 agents with 0 interval between

requests, the the only result was, that the response time rose to 9,3 s and the throughput (number of requests served per second) was very unstable, but server was still able to respond to all requests properly.

In order to be able to compare the performance of both servers under the load testing, the Windows server was also put under the test of 20 agents with constant requests for 10 minutes. During that test the SQL server was taking 52% of CPU time on average, whereas *w3wp.exe* was taking 34%.



Graph 10: Response times of Windows server for single agent

In Table 19 there is a comparison of the both server response times and throughput (requests served per second) when tested with no interval between the requests.

	Single agent		**20 agents**	
	Linux	**Windows**	**Linux**	**Windows**
Average response time [s]	0,183	0,030	2,624	0,236
Response time standard deviation	0,435	0,020	7,021	0,100
Average throughput	5,367	32,333	7,593	83,537
Throughput standard deviation	3,589	2,348	11,107	10,267

Table 19: Comparison of environments under load test by single agent

6 Conclusion

There are two conclusions to be drawn from the comparison presented in this paper. First is, that Mono is under dynamic development and that its implementation of 2.0 version of ASP.NET framework works very well if it comes to compatibility. The availability and relative ease of use of the database connectors, capability to work with numerous browsers and on variety of operating systems makes the public more and more aware of its existence.

The information system developed for testing purposes of this paper however shows a significant performance drawbacks of the combination of Apache web server, MySQL database and ASP.NET framework implementation by Mono. From the tests performed in chapter 5 , it is clear, that the most significant problem lies in database. When only small tables were addressed, the open source solution was comparable with the Microsoft's original.

MySQL is famous for its MyISAM database engine, which was not used, because it is non-transactional, whereas Microsoft SQL's engine is and though transactions were not used, the comparison tried to be held on possibly equally equipped environments. For an experiment however, the database in MySQL was recreated with MyISAM engine and few tests were run on it. The results were even worse, therefore the outcome is, that presented ASP.NET web pages hosting solution for Linux environment is not suitable for situations where composite queries have to be run upon large tables. Presented solution has also not proven itself well in situations of higher traffic. With those facts in mind, the meaningful application of the solution limits to the home and possibly small business use, especially if such place already has Linux server using Apache and/or MySQL.

Furthermore, there is a possible licensing problem that has been pointed out by Free Software Foundation [15]. Microsoft's Community Promise does not guarantee, that it is not going to be revoked and in such case, all open source C# implementations and all software written in those would become illegal. Therefore production use of Mono is tied with some risks.

7 References

[1] *ASP.NET - Mono* [online]. ©, last revision 2010-03-25 [quoted 2010-03-29]. <http://www.mono-project.com/ASP.NET>.

[2] *Database Access - Mono* [online]. ©, last revision 2009-12-03 [quoted 2010-03-29]. <http://www.mono-project.com/Database_Access>.

[3] BREHM, T. - TIMME, F.. *The Perfect Server - OpenSUSE 11.1 [ISPConfig 3]* [online]. ©2009, last revision 2009-03-18 [quoted 2010-03-11]. <http://www.howtoforge.com/perfect-server-opensuse-11.1-ispconfig-3>.

[4] *February 2010 Web Server Survey - Netcraft* [online]. ©2010, last revision 2010-02-22 [quoted 2010-03-18]. <http://news.netcraft.com/archives/2010/02/22/february_2010_web_server_survey.html>.

[5] *Mod mono* [online]. ©2010, last revision 2010-01-17 [quoted 2010-06-09]. <http://www.mono-project.com/Mod_mono>.

[6] Adam Farnik. *Mod mono on Apache* [online]. ©2010, last revision 2010-03-22 [quoted 2010-03-22]. <http://en.opensuse.org/Mod_mono_on_Apache>.

[7] *Server Core Installation Option Getting Started Guide* [online]. ©2008, last revision 2009-10-22 [quoted 2010-03-24]. <http://technet.microsoft.com/en-us/library/cc753802%28WS.10%29.aspx>.

[8] *Hardware and Software Requirements for Installing SQL Server 2008* [online]. ©2008, last revision 2009-08-31 [quoted 2010-03-25]. <http://msdn.microsoft.com/en-us/library/ms143506.aspx>.

[9] MAXIA, Giuseppe. *Sample database with test suite in Launchpad* [online]. ©2008, last revision 2009-03-29 [quoted 2010-04-01]. <https://launchpad.net/test-db/>.

[10] *MySQL 5.1 Reference Manual :: 21.2 MySQL Connector/NET* [online]. ©2010, last revision [quoted 2010-04-13]. <http://dev.mysql.com/doc/refman/5.1/en/connector-net.html>.

[11] FARNIK Adam. *Bug 609478 - Validating control inside Repeater prevents ItemCommand to fire* [online]. ©2010, last revision 2010-05-31 [quoted 2010-06-05]. <https://bugzilla.novell.com/show_bug.cgi?id=609478>.

[12] LAMPING, U. - SHARPE, R. - WARNICKE, E.. *Wireshark User's Guide* [online]. ©2004-2010, last revision [quoted 2010-06-23]. <http://www.wireshark.org/docs/wsug_html_chunked/>.

[13] *Selenium Documentation* [online]. ©2009, last revision 2010-06-20 [quoted 2010-06-22]. <http://seleniumhq.org/docs/>.

[14] GOLDBERG, Corey. *Pylot | Open Source Web Performance Tool* [online]. ©2007-2009, last revision 2010-02-26 [quoted 2010-06-23]. <http://www.pylot.org/>.

[15] STALLMAN, Richard M.. *Why free software shouldn't depend on Mono or C#* [online]. ©2009, last revision 2009-12-11 [quoted 2010-06-23]. <http://www.fsf.org/news/dont-depend-on-mono>.

8 Annexes

CD content

- Information system source files for Windows
- Information system source files for Linux
- DAL dataset
- Employees database for MySQL
- Table scripts for MS SQL
- Stored procedures code for MS SQL
- Stored procedures code for MySQL
- Selenium test suite
- Pylot test cases

I. MS SQL table scripts

```
CREATE TABLE [dbo].[employees](
      [emp_no] [int] IDENTITY(10001, 1) NOT NULL,
      [birth_date] [date] NOT NULL,
      [first_name] [varchar](14) NOT NULL,
      [last_name] [varchar](16) NOT NULL,
      [gender] [char](1) NOT NULL,
      [hire_date] [date] NOT NULL,
      CONSTRAINT [PK_employees] PRIMARY KEY CLUSTERED
            ([emp_no] ASC),
      CONSTRAINT [CK_employees] CHECK  (([gender]='M' OR
            [gender]='F'))
)

CREATE TABLE [dbo].[departments](
      [dept_no] [char](4) NOT NULL,
      [dept_name] [varchar](40) NOT NULL,
      CONSTRAINT [PK_departments] PRIMARY KEY CLUSTERED
            ([dept_no] ASC)
)

CREATE TABLE [dbo].[dept_emp](
      [emp_no] [int] NOT NULL,
      [dept_no] [char](4) NOT NULL,
      [from_date] [date] NOT NULL,
      [to_date] [date] NOT NULL,
      CONSTRAINT [PK_dept_emp] PRIMARY KEY CLUSTERED
      (
            [emp_no] ASC,
            [dept_no] ASC
      )
)

CREATE TABLE [dbo].[dept_manager](
      [dept_no] [char](4) NOT NULL,
      [emp_no] [int] NOT NULL,
      [from_date] [date] NOT NULL,
      [to_date] [date] NOT NULL,
      CONSTRAINT [PK_dept_manager] PRIMARY KEY CLUSTERED
      (
            [dept_no] ASC,
            [emp_no] ASC
      )
)
```

```
CREATE TABLE [dbo].[salaries](
      [emp_no] [int] NOT NULL,
      [salary] [int] NOT NULL,
      [from_date] [date] NOT NULL,
      [to_date] [date] NOT NULL,
      CONSTRAINT [PK_salaries] PRIMARY KEY CLUSTERED
      (
            [emp_no] ASC,
            [from_date] ASC
      )
)

CREATE TABLE [dbo].[titles](
      [emp_no] [int] NOT NULL,
      [title] [varchar](50) NOT NULL,
      [from_date] [date] NOT NULL,
      [to_date] [date] NULL,
      CONSTRAINT [PK_titles] PRIMARY KEY CLUSTERED
      (
            [emp_no] ASC,
            [title] ASC,
            [from_date] ASC
      )
)
```

II. MS SQL stored procedures code

```
CREATE PROCEDURE [dbo].[GetEmployeesRowCount]
(
  @searchName varchar(14),    -- name to search for
  @searchSurname varchar(16), -- surname to search for
  @searchDept char(4)         -- department code to search for
)
AS

DECLARE @sql nvarchar(1000)  -- string to compose SQL query in

SET @sql = 'SELECT COUNT(*) FROM employees'

IF LEN(@searchDept) > 0
BEGIN
  SET @sql = @sql + ' INNER JOIN dept_emp ON
  employees.emp_no = dept_emp.emp_no
  WHERE (dept_emp.dept_no = ''' + @searchDept + ''')'
  IF LEN(@searchName) > 0
    SET @sql = @sql + ' AND (employees.first_name = '''
      + @searchName + ''')'
```

```
  IF LEN(@searchSurname) > 0
    SET @sql = @sql + ' AND (employees.last_name = '''
      + @searchSurname + ''')'
END
ELSE
BEGIN
  IF LEN(@searchName) > 0
  BEGIN
    SET @sql = @sql + ' WHERE (employees.first_name = '''
      + @searchName + ''')'
    IF LEN(@searchSurname) > 0
      SET @sql = @sql + ' AND (employees.last_name = '''
        + @searchSurname + ''')'
  END
  ELSE
    IF LEN(@searchSurname) > 0
      SET @sql = @sql + ' WHERE (employees.last_name = '''
        + @searchSurname + ''')'
END

EXEC sp_executesql @sql



CREATE PROCEDURE [dbo].[GetEmployeesPaged]
(
  @startRowIndex int,
  @maximumRows int,
  @sortExpression varchar(20),
  @searchName varchar(14),
  @searchSurname varchar(16),
  @searchDept char(4)
)
AS

DECLARE @sql nvarchar(4000)  -- string to compose SQL query in

--- COMPOSING BEGINING OF QUERY ---
SET @sql = 'SELECT emp_no, birth_date, first_name, last_name,
  gender, hire_date FROM ('

                    --- INNER QUERRY ---
    + 'SELECT employees.emp_no, birth_date, first_name,
      last_name, gender, hire_date, ROW_NUMBER() OVER
      (ORDER BY employees.'
```

```
--- ADDING SORTING PARAMETERS ---
IF LEN(@sortExpression) > 0
  SET @sql = @sql + @sortExpression
ELSE
  SET @sql = @sql + 'emp_no'

--- CONTINUING INNER QUERY ---
SET @sql = @sql + ') AS row_no FROM employees'

--- ADDING SEARCH PARAMETERS ---
IF LEN(@searchDept) > 0
BEGIN
  SET @sql = @sql + ' INNER JOIN dept_emp ON employees.emp_no
    = dept_emp.emp_no WHERE (dept_emp.dept_no = '''
      + @searchDept + ''')'
  IF LEN(@searchName) > 0
    SET @sql = @sql + ' AND (employees.first_name = '''
      + @searchName + ''')'
  IF LEN(@searchSurname) > 0
    SET @sql = @sql + ' AND (employees.last_name = '''
      + @searchSurname + ''')'
END
ELSE
BEGIN
  IF LEN(@searchName) > 0
  BEGIN
    SET @sql = @sql + ' WHERE (employees.first_name = '''
      + @searchName + ''')'
    IF LEN(@searchSurname) > 0
      SET @sql = @sql + ' AND (employees.last_name = '''
        + @searchSurname + ''')'
  END
  ELSE
    IF LEN(@searchSurname) > 0
      SET @sql = @sql + ' WHERE (employees.last_name = '''
        + @searchSurname + ''')'
END

--- FINISHING QUERY ---
SET @sql = @sql + ') AS numbered_employees WHERE row_no
  BETWEEN '
  + CONVERT(varchar(10), @startRowIndex) + ' + 1 AND '
  + CONVERT(varchar(10), @startRowIndex) + ' + '
  +  CONVERT(varchar(10), @maximumRows)

EXEC sp_executesql @sql
```

III. Pylot test cases definition

```
<testcases>

 <case>
  <url>http://140.134.23.133</url>
  <method>GET</method>
  <verify>Home page</verify>
 </case>

 <case>
  <url>http://140.134.23.133/Employee.aspx?emp_no=34289</url>
  <method>GET</method>
  <verify>Boguraev</verify>
  <verify>Yoshinari</verify>
 </case>

 <case>
  <url>http://140.134.23.133/Employee.aspx?emp_no=26754</url>
  <method>GET</method>
  <verify>Kowalchuk</verify>
  <verify>Lucien</verify>
 </case>

 <case>
  <url>http://140.134.23.133/Employee.aspx?emp_no=96267</url>
  <method>GET</method>
  <verify>Schrift</verify>
  <verify>Kazuhiro</verify>
 </case>

 <case>
  <url>http://140.134.23.133/Employee.aspx?emp_no=428861</url>
  <method>GET</method>
  <verify>Delgrossi</verify>
  <verify>Luigi</verify>
 </case>

 <case>
  <url>http://140.134.23.133/Employee.aspx?emp_no=66209</url>
  <method>GET</method>
  <verify>Schnelling</verify>
  <verify>Yinghua</verify>
 </case>

 <case>
  <url>http://140.134.23.133/Employee.aspx?emp_no=219005</url>
  <method>GET</method>
```

```
  <verify>Bolotov</verify>
  <verify>Mircea</verify>
 </case>

 <case>
  <url>http://140.134.23.133/EmployeeList.aspx</url>
  <method>GET</method>
  <verify>10001</verify>
  <verify>10020</verify>
 </case>

 <case>
  <url>http://140.134.23.133/NewEmployee.aspx</url>
  <method>GET</method>
 </case>

</testcases>
```